UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

BACHELOR IN COMPUTER SCIENCE

Intelligent system based on mixed reality for diagnosis and assistance in primary care

Pablo del Hoyo Abad

July, 2024

INTELLIGENT SYSTEM BASED ON MIXED REALITY FOR DIAGNOSIS AND ASSISTANCE IN PRIMARY CARE

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Department of Technology and Information Systems

BACHELOR IN COMPUTER SCIENCE

COMPUTING

Intelligent system based on mixed reality for diagnosis and assistance in primary care

Author: Pablo del Hoyo Abad

Advisor: David Vallejo Fernández

Co-advisor: Francisco Manuel García Sánchez-Belmonte

July, 2024

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

Abstract

The use of new technologies is what characterises a new approach to healthcare known as smart healthcare. By employing the tools provided by Artificial Intelligence (AI), Mixed Reality (MR), the Internet of Things or cloud computing, healthcare systems have been able to diagnose and treat diseases more efficiently. At the core of those kinds of improvements are intelligent clinical Decision Support Systems (DSSs,) which aids physicians in the diagnosis and treatment process.

In Spain, primary care attention faces many challenges. One of them is the integration of technological advancements which could improve the quality of the services provided by medical staff. It is in this context in which a system which merges two fields with the potential of having a profound impact in medicine such as AI and MR has been developed as part of this project.

The project consisted in the development of a system employing methods and techniques from AI and MR to enhance the diagnosis and treatment capabilities of primary care staff. At the core of the system, there is a MR virtual assistant which indicates the professional the steps they have to perform according to a medical protocol which has had to be specified previously in a web application also develop as part of this project. Additionally, the system supports querying a Large Language Model (LLM) about the contents of documents containing relevant information for a medical procedure. Given the potential sensitivity of the documents which the LLM uses to extract the information, the LLM has been hosted on a server for compute intensive workloads owned by the UCLM.

Apart from helping the professional in the task of diagnosing and treating a disease, the system can also be used by medical students to put in practice their knowledge without the constant supervision of an expert who corrects them. Finally, the system may easily be adapted to areas different than medicine where procedures are clearly defined, such as the ones involving the repair of products.

Resumen

El uso de las nuevas tecnologías es lo que caracteriza un nuevo enfoque de la asistencia sanitaria conocido como salud inteligente. Mediante el empleo de las herramientas proporcionadas por la inteligencia artificial, la realidad mixta, el Internet de las Cosas o la computación en la nube, los sistemas sanitarios han podido diagnosticar y tratar enfermedades de forma más eficientemente. La base de ese tipo de mejoras se encuentran en los sistemas clínicos de soporte a la decisión, los cuales ayudan a los médicos en el proceso de diagnóstico y tratamiento de enfermedades.

En España, la atención primaria se enfrenta a numerosos retos. Uno de ellos es la integración de avances tecnológicos que puedan mejorar la calidad de los servicios prestados por el personal médico. Es en este contexto en el que un sistema que aúna dos campos con el potencial de tener un profundo impacto en la medicina, como son AI y MR, ha sido desarrollado el presente proyecto.

El proyecto consistió en el desarrollo de un sistema que emplea métodos y técnicas de inteligencia artificial y realidad mixta para mejorar las capacidades de diagnóstico y tratamiento del personal de atención primaria. Este se basa en un asistente virtual que indica al profesional los pasos que tiene que realizar según un protocolo médico que ha tenido que ser especificado previamente en una aplicación web, también desarrollada como parte de este proyecto. Además, el sistema permite consultar a un LLM sobre el contenido de los documentos que contienen información relevante para un procedimiento médico. Dada la posible sensibilidad de los documentos que el LLM usa para extraer la información, el modelo se ha desplegado en un servidor propiedad de la UCLM.

Además de ayudar al profesional en la tarea de diagnosticar y tratar una enfermedad, el sistema también puede ser utilizado por estudiantes de medicina para poner en práctica sus conocimientos sin la supervisión constante de un experto que les corrija. Finalmente, el sistema podría adaptarse fácilmente a otros ámbitos distintos de la medicina en los que los procedimientos están claramente definidos, como por ejemplo los que implican la reparación de productos.

Acknowledgement

En primer lugar, quiero expresar mi más sincero agradecimiento a mi familia, quienes han sido mi mayor apoyo a lo largo de todo este camino. Gracias a mis padres, por su amor incondicional, su paciencia y por haberme brindado siempre las mejores oportunidades. A mis hermanos, por ser una fuente constante de ánimo y por creer en mí en cada paso que he dado.

A mis profesores del colegio donde me he formado desde los tres años hasta terminar el Bachillerato con dieciocho. La influencia que he recibido de ellos ha determinado en gran medida la persona que soy hoy en día.

Una mención especial merecen mis tutores de TFG, David y Paco. Su orientación, confianza, compromiso y, sobre todo, paciencia, han sido cruciales para la realización de este trabajo.

Finalmente, quiero agradecer a todos mis amigos y compañeros de estudios, quienes han sido una parte esencial de esta experiencia.

Pablo

A mi familia

Contents

List of Tables

List of Figures

List of code listings

List of acronyms

ADS	Adaptive Software Development
AGI	Artificial General Intelligence
AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
ASGI	Asynchronous Server Gateway Interface
BMI	Body Mass Index
CA	Certificate Authority
CGI	Common Gateway Interface
CLI	Command Line Interface
CRUD	Create Read Update Delete
CSS	Cascading Style Sheets
DNN	Deep Neural Network
DOM	Document Object Model
DSL	Domain Specific Language
DSS	Decision Support System
ER	Entity–Relationship
GPU	Graphics Processing Unit
HMD	Head Mounted Display
HTML	HypertText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LLM	Large Language Model
MIME	Multipurpose Internet Mail Extensions
MRTK	Mixed Reality Tool Kit
MR	Mixed Reality
NIH	National Institutes of Health
NLP	Natural Language Processing
ORM	Object Relational Mapping
OS	Operating System
PC	Personal Computer

RAG	Retrival Augmented Generation
RDBMS	Relational Database Management System
RSC	React Server Components
SEO	Search Engine Optimization
SPA	Single Page Application
SSE	Server Side Events
SSR	Server Side Rendering
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
VPN	Virtual Private Network
VPS	Virtual Private Server
VR	Virtual Reality
XP	Extreme Programming
XR	Extended Reality

Chapter 1

Introduction

In this chapter, an introduction to the project is given. Firstly, the concept of primary health and its origin are discussed. Then, the new paradigm of smart healthcare is described, with a focus on Artificial Intelligence (AI) and Mixed Reality (MR). Finally, the project proposal and the structure of this document is detailed.

1.1 Primary health care

Primary health care is a cornerstone of a robust and equitable health system, serving as the first point of contact for individuals and families within the healthcare system. It includes a broad spectrum of health services including prevention, wellness, and treatment for common illnesses and conditions. Primary health care aims to improve health outcomes, reduce health disparities, and enhance the efficiency of healthcare delivery [Org23].

The concept of primary health care gained global recognition with the Alma-Ata Declaration in 1978, which emerged from an international conference organized by the World Health Organization (WHO) and UNICEF. This declaration emphasized that health is a fundamental human right and that governments have a responsibility to ensure the health of their people through accessible, affordable, and equitable primary health care services [HM08].

1.2 Smart healthcare

In order to make the principles outlined in the Alma-Ata declaration possible, the healthcare sector has found in new technologies a great ally. The use of information technologies in healthcare has given rise to the concept of smart healthcare [TYLG+19]. Smart healthcare is characterised by the use of technological advancements such as big data, Artificial Intelligence (AI), the Internet of Things, 5G, and cloud computing to provide a more efficient and personalised way of treating patients. From the perspective of the patient, this new paradigm has made possible to have a constant monitoring of their health by using wearable devices, receive assistance thanks to the use of virtual assistants or the possibility of contacting health care professionals with the help of telemedicine solutions. On the other hand, physicians are also benefiting from this new approach to healthcare. Specifically, intelligent clinical Decision Support Systems (DSSs) have made the diagnosis more accurate. As a consequence,

the patient condition can be described more precisely and personalized treatments become more effective.

The area of Computer Science at the core of those DSSs is AI. In some situations, these kind of systems have made better predictions than human doctors [DR15]. Specifically, machine learning models have surpassed the diagnosis capabilities of physicians in some experiments, specially those related to pathology and imaging [WS12]. However, the decisions taken by these kind of systems are not blindly applied but serve as a complement to the judgments made by a human doctor.

Another technology which has had a great impact in healthcare is Mixed Reality (MR) [TYLG+19]. MR allows integrating virtual objects into the real environment without having to resort to the cumbersome devices and external navigation systems which characterize most Augmented Reality (AR) solutions [HFS+19]. This has enabled physicians to get a more effective training by testing their knowledge and skills in an environment almost identical to the real one without the costs and ethical barriers real experimentation pose [SSC+18]. MR does not only implicitly benefit patients by being attended by well trained doctors but can be used to reduce the gap in knowledge between both of them. Given the visualization and interaction possibilities MR offers, a doctor can support their explanations with 3D models which are simultaneously seen by the patient. This mechanism has led to a more accurate and simple communication between both of them [HNM16]. Additionally, MR has been used to make telemedicine more convenient and intuitive compared to the classical 2D screen [HFS+19].

Figure 1.1: Visual representation of a healthcare professional making use of MR technology to assist a patient

1.3 Context

In Spain, the Alma-Ata declaration served as the basis for the *Real Decreto 137/1984 de Estructuras Básicas de Salud*[1] which laid the foundations of the Spanish primary healthcare system. In a study published by The European Observatory on Health Systems and Policies [KBH+15], the Spanish primary care system was regarded as one of the best in the several dimensions which were analyzed. However, even though Spaniards have a positive opinion about it, specially when they are asked about the assistance received by the professionals, overall, its image has been affected as a consequence of the budget cuts which it has experienced [LC17].

The primary healthcare system faces several challenges. Apart from the underfunding that the system still suffers, the incorporation of new technological advancements which could help medical staff as well as patients must also be addressed [LC17].

Given the transformative effects MR and AI can have in medicine, the system developed as part of this project merges both fields with the aim of enhancing the diagnosis capabilities of medical staff. This is done in an effort to incorporate new technologies to the healthcare system. In order to ease its adoption, MR hardware affordable for consumer budgets has been employed.

Notice that even though the system is focused on the healthcare sector, the techniques which have been employed to merge AI and MR could be adapted to create similar applications for other kinds of fields. One which could take advantage of the methods this project introduces is the construction sector. Specifically, in a future iteration of the system, the virtual assistant could help a builder to correctly place the materials by showing them virtual materials at the correct location. After the builder had put, for example, a brick where the assistant had told them, that event would be detected by the system and it would move to the next step. This would relieve the foreman from having to constantly supervise the builders and would enable them to focus on other task related to the construction site.

Additionally, given the assistant's knowledge on the specific domain and its abilities to detect how the person has reacted to a instruction given by it, it could help students to test their knowledge in a practical setting without the scrutiny of an expert because its role could be almost substituted by the virtual assistant. Furthermore, data related to the way a student has followed a specific instruction could be gathered and analyzed to evaluate their performance.

1.4 Project proposal

The project detailed in this document consist in the design and implementation of a system which makes use of Artificial Intelligence (AI) and Mixed Reality (MR) to enhance the

[1] `https://www.boe.es/eli/es/rd/1984/01/11/137/con`

diagnosis and treatment capabilities of primary care staff. By making use of a MR headset, doctors and nurses will be able to follow the steps indicated by an intelligent assistant to perform the diagnosis and apply the right treatment with the advantage of not having to utilize any device which limit the actions they can do with their hands. To ensure medical protocols provided by the assistant are correct, these are previously specified in a web application also developed as part of the project. These medical procedures will be represented as flowcharts, which indicate the next step the assistant will show based on the options chosen previously.

Nevertheless, accurately specifying a medical procedure is an arduous task which requires the precious time of an expert. In case no one is available, a thorough study of the domain is necessary. That implies reading and assimilating a considerable amount of technical documentation. Instead of waiting until the medical procedure is fully specified, a Large Language Model (LLM) can be queried for the pieces of information which still are not shown by the assistant. The LLM could also be used to provide more context about specific concepts or facts which were not deemed necessary to include in a step. The model would obtain the needed information to answer a query from a set of documents, which have to be uploaded previously using the web application.

The medical care staff must be aware of the trade-offs between utilizing the assistant and LLM. The assistant is guaranteed to be correct as long as the procedure does not contain any errors. However, it is more inflexible given the constraint of having to structure the information in the form of a flowchart. Instead, the LLM can process documentation written in natural language so, in that sense, it is more flexible. However, its reasoning capabilities are not comparable to the ones humans have so there is the possibility that it outputs incorrect information. The use of one does not preclude the interaction with the other so both can be used during the execution of a medical procedure.

Given the sensitivity of the data which the LLM may see to obtain the knowledge, the model will be hosted on a server for compute intensive workloads owned by the UCLM.

A depiction of the way a person interacts with the system is shown in figure 1.2.

1.5 Applications

Medical staff can greatly benefit from the system developed as part of this project for several reasons. Firstly, the system would reduce the number of mistakes a professional can make. Those errors are normally a result of the stress and tiredness they experience because of their responsibility and high number of hours the sometimes have to spent at their workplace. The system could be used to ensure that specific medical guidelines have been followed correctly, thus making the diagnosis process more accurate. On the other hand, by using MR, the system is completely integrated into their environment. As a consequence, the professional does not have to worry about using devices which could limit the range of

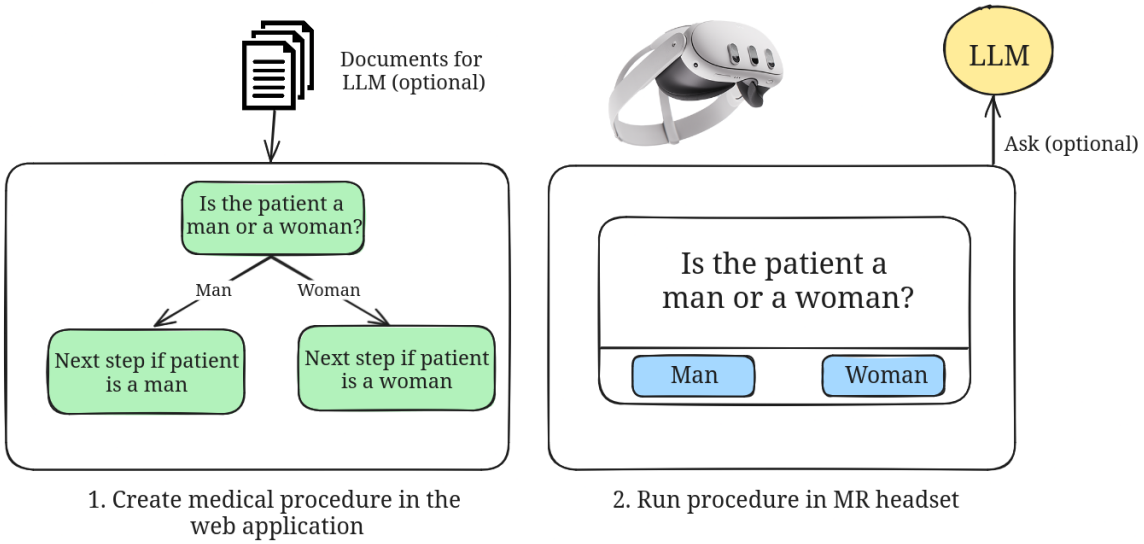

Figure 1.2: Interaction with the developed system

actions and movements which might be necessary during a diagnosis process because they only need to wear a MR headset. Additionally, the environment could influence the decisions taken by the AI component which is part of the system.

1.6 Document structure

The rest of this document has been structured into the chapters listed below, according to the standards for Final Degree Projects of the *School of Computer Science*, of the *Castilla-La Mancha University* (UCLM).

Chapter 2: Objectives

In this chapter, the objectives and subgoals of the project are outlined.

Chapter 3: State of art

This chapter gives an overview of the current state of the main computer science fields which this project has benefited from.

Chapter 4: Methodology

In this chapter, the methodology which has been followed is described. It also includes a section on the resources which have been utilized for the development of the project.

Chapter 5: Architecture

This chapter describes the architecture of the developed system. Apart from outlining how the system is structured, alternative approaches and major difficulties are also discussed.

Chapter 6: Results

This chapter presents the final results of the project.

Chapter 7: Conclusions

This chapter discusses whether the proposed objectives have been achieved. Additionally, it proposes future lines of work.

Chapter 2

Objectives

In this chapter, the objectives of the project are outlined. Specifically, the general objective is explained and then the subgoals which supports the main objective are detailed.

2.1 General objectives

The goal of the project is the design and development of a system based on Mixed Reality (MR) and Artificial Intelligence (AI) to enhance the diagnostic and treatment capabilities of primary care medical staff. This will be possible by creating a web application in which procedures commonly carry out in medicine, which are referred as medical protocols, are specified. The execution of a protocol in a MR application running on the Meta Quest 3 headset will support asking a Large Language Model (LLM) specialized on the contents of the procedure with the aim of recieving answers with relevant information about it. Given the sensitivity of the information the LLM may have to see in order to gain that knowledge, the model will be hosted on a server for compute intensive workloads owned by the university.

2.2 Specific objectives

The specific objectives which supports the general one are the following:

- **Detailed study of Mixed Reality devices.** The Mixed Reality paradigm will be studied as well as the devices which make it possible. Additionally, the tools which are used in the development of a MR applications will be researched.
- **Detailed study of large language models.** LLMs will be studied to know the best way in which to fit them into the project. Additionally, the tools and technologies which make it possible to run these kinds of models in consumer hardware will be analyzed.
- **Design and development of a MR application that allows the execution of medical protocols by primary care staff**. An application for the Meta Quest 3 headset will be developed which will execute medical protocols by employing the unique features the MR paradigm offers. It will also allow interacting with the LLM in an appropiate manner.

- **Design and development of a web application that allows the definition of medical protocols and the association of information basis for the inference process.**. A web application will be developed in which medical protocols will be defined. Moreover, it will allow attaching to each one the information which will be used by the LLM to answer a question when a protocol is being executed.
- **Deployment of a real medical protocol which allows the diagnosis and treatment of a disease.** A real medical protocol will be defined with the web application and will be executed on the MR headset to prove that the system can easily integrate already established medical protocols.

Chapter 3

State of art

In this chapter, an overview of the current state of the main computer science fields which this project has benefited from is given. These are deep learning, web development and Extended Reality (XR).

3.1 Deep learning

The most recent advances in Artificial Intelligence (AI) have been possible thanks to the techniques develop by the deep learning community. Deep learning is the subfield of AI which uses a kind of model, known as Deep Neural Network (DNN) [GBC16], as its main tool to approach different problems. One field where DNNs have achieved state of the art results is in Natural Language Processing (NLP) [TSK+20]. Specifically, Large Language Models (LLMs) have shown very promising results in tasks such as text summarization, question answering, text classification or text generation [HQS+23].

3.1.1 Supervised learning

Some kind of problems can be approached using the supervised learning framework. In this kind of problems, you have some input data and its corresponding output and the objective is to create a function which maps the input data to the output data. That mapping is obtained by feeding a model those input-outputs pairs, which will adjust its parameters during the learning process to capture the relationships between the inputs and the outputs.

In order to more formally define these problems, we will use a branch of probability theory known as *statistical learning theory* [HTFF09]. Let X_s be the set containing the input data and Y_s be the output data and suppose there exists a probability distribution $p(x,y)$ over X_s and Y_s from which we have sampled to obtain the training data $D = \{(x_1,y_1),...,(x_n,y_n)\}$. Our objective is to find a function $h : X_s \to Y_s$, known as the hypothesis, for which, given a real valued function $l(\hat{y},y)$ which measures the dissimilarity between its arguments, the value $E_{(X,Y)\sim p}[l(h(X),Y)]$ is minimized. Since we don't have access to the distribution, the expectation is usually approximated by the following expression [Kar16]:

$$L = \frac{1}{|D|} \sum_{(x,y)\in D} l(h(x),y) \tag{3.1}$$

And that expression is the objective function for the optimization problem you setup to try to find $h(x)$.

There does not exist a rule which tells which dissimilarity function $L(\hat{y}, y)$, which is known as the loss function, to use. Sometimes you choose one for convenience and because it serves its purpose of measuring dissimilarity but the most common thing is to choose one based on some statistical property we are interested in. Specifically, there is a branch of statistics called estimation theory which proposes different methods to estimate the parameters of a statistical model. Among them, one which is particularly used in deep learning is called maximum likelihood estimation [Cha21]. Therefore, the loss function is chosen so that finding the parameter configuration which minimizes it is equivalent to maximizing the likelihood function.

The task described above would be easier if we restrict the choice of $h(x)$ to a set of functions with a specific form [Kar16]. In deep learning, we make use of a kind of functions which were initially inspired by a very simple model of the brain and, as a consequence, they were named neural networks. It is important to remember that the objective of neural networks is not to create a perfect model for the brain, but a *machine* which achieve statistical generalization [GBC16]. However, there is a surprising mathematical fact about them and it is that they are universal function approximators. They are able to approximate functions of some class (and that *class* of functions is enough for the objectives we have in a supervised learning problem) to an arbitrary degree of precision, provided the neural network is "big" enough [GBC16]. The hardest part is finding the right configuration of the model parameters, a process known as training, which makes that happen. Neural networks not only work because of that but also because they generalize very well for data outside of the training dataset with the help of some techniques to prevent the model from memorizing the data. The reason why this happen is not deeply understood.

The simplest neural network is called feedforward neural network. The model is the following [ZLLS23, GBC16]. Let $\boldsymbol{x} \in \mathsf{R}^d$ be a vector which contains the features for a particular data point. Let $\boldsymbol{W} \in \mathsf{R}^{d \times h}$ be matrix, called the weight matrix and let $\boldsymbol{b} \in \mathsf{R}^h$ be a vector called the bias. Assuming $\phi(\boldsymbol{x})$ is a nonlinear function applied elementwise, which is known as the activation function, the following transformation gives raise to the vector $\boldsymbol{o} \in \mathsf{R}^m$, which represent the hidden units for this particular layer:

$$\boldsymbol{o} = \phi(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}) \tag{3.2}$$

The output $\boldsymbol{o}$ can be used as an input to another transformation which follows the same structure as the one in 3.2, but with a different weight matrix and bias, whose dimension must be set appropriately so that the matrix multiplication and addition can be applied. The output dimension of that vector does not have to be equal to the input dimension and, in fact,

it will normally be different. Neural networks become deep when you stack a lot of these transformations, one after another. The number of transformation you need will depend on the task at hand and the only way to find one which works well is by trying different numbers.

The number of transformations you stack, which is referred in deep learning jargon as the number of layers of the neural network [GBC16], is an example of a hyperparameter because it is not learned from the data but it is fixed when you choose the neural network architecture. Another example of a hyperparameter is the dimension of the hidden units.

A graphical representation of a feedforward neural network with two hidden layers is shown in figure 3.1.

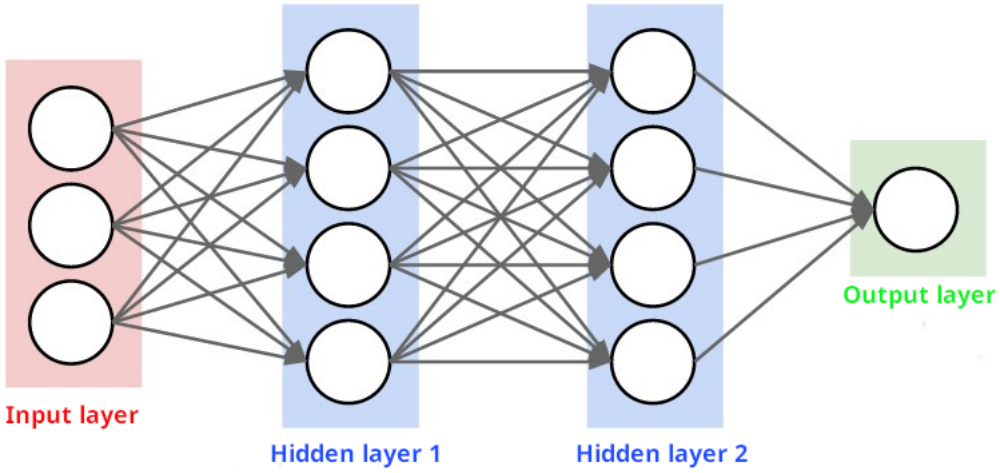

Figure 3.1: Graphical representation of a feedforward neural network
Image from `https://cs231n.github.io/neural-networks-1/`

The nonlinear function $\phi(x)$ is what allows the neural network to learn nonlinear mappings. The ones which were first used were the sigmoid, $\sigma(x) = \frac{1}{1+\exp(-x)}$ and $\tanh(x)$. However, they have some undesirable properties which make the training process hard. For that reason, other nonlinear function like the rectified linear unit or ReLU, $\max(0,x)$, are used [GBC16].

Finally, notice that our training set consists of several features vectors. Assume we have n. We could represent it using the matrix $\boldsymbol{X} \in \mathsf{R}^{d \times n}$, where the i^{th} columns corresponds to the i^{th} feature vector. This allows us to calculate the transformation 3.2 for all the feature vectors in the following way

$$\boldsymbol{O} = \phi(\boldsymbol{W}^T \boldsymbol{X} + \boldsymbol{B}) \tag{3.3}$$

and as a consequence, $\boldsymbol{O} \in \mathsf{R}^{h \times n}$ and $\boldsymbol{B} \in \mathsf{R}^{h \times n}$. $\boldsymbol{B}$ results from horizontally stacking the column vector $\boldsymbol{b}$ n times. This particular way of applying the transformation allows us to leverage the parallel operations at which excel at.

3.1.2 Training neural networks

The process of training a neural network consists in finding the parameter configuration which results in the smallest loss evaluated over the training set. In the case of a feedforward neural network, these parameters are the weight matrices and the bias vectors. We will refer to those trainable parameters as $\boldsymbol{\theta} \in \mathsf{R}^p$, where p is the number of parameters. Nevertheless, that is not the ultimate goal. The real focus must be on achieving a good performance on data which has not been seen during training.

Considering all the optimization algorithms that exist, the ones used to train a neural network are remarkably simple because what is required from the objective function, which in this case is the loss function over the training set, $L : \mathsf{R}^p \to \mathsf{R}$, is its gradient with respect to the model parameters. Let's denote that value by $\nabla_{\boldsymbol{\theta}} L$,

One of the most straightforward optimization algorithm is gradient descent [ZLLS23]. Gradient descent updates the parameters using the following rule:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} L \tag{3.4}$$

where $\eta > 0$ is a hyperparameter called the learning rate. Other algorithms use other update rules but, in all of them, the gradient is normally involved.

The problem of using the gradient is that you need to iterate over all the training samples in order to calculate it for a specific parameter configuration and, as a consequence, if you have a lot of data, it will take a considerable amount of time to apply a parameter update. Since this optimization process is actually a proxy to achieve generalization, researchers thought an estimate of the full gradient would be enough for the task of training a neural network. The idea is to randomly choose a certain amount of training samples, use only those to calculate the gradient and use it in the update rule. And surprisingly, it worked [ZLLS23]. In deep learning, the number of training samples we use to estimate the gradient is called the batch size and it is also considered a hyperparameter. Let's denote the loss over those training samples, known in deep learning as a mini-batch, as L_b

In gradient descent, when we use an estimate of the gradient instead of the full the gradient, the algorithm is known as stochastic gradient descent.

3.1.3 Calculating the gradients

Since a neural network is a composition of differentiable functions [1], the gradient $\nabla_{\boldsymbol{\theta}} L_b$ can be calculated using the multivariable chain rule. There is an algorithm for feedforward neural networks, known as the backpropagation algorithm, which uses the multivariable chain rule to efficiently calculate it [GBC16]. However, the backpropation algorithm only

[1] Actually, ReLU is not differentiable at $x = 0$ but in practice it does not matter because the floating point representation makes it very unlikely that we will have in memory *exactly* 0

works for that kind of functions. There are other kinds of neural networks which use other transformations than 3.2 and, as a consequence, some modifications to the backpropagation algorithm are required to take them into account. Those modifications were obtained by applying the differentiation rules to those new transformations and explicitely incorporating them to the code in charge of calculating the gradient.

However, with the advent of automatic differentiation frameworks like PyTorch, Tensorflow or JAX, you do not need to worry about updating the code which calculates the gradient when you add or delete some kind of operation to the neural network architecture because those frameworks transparently handle all of that for you. What they all do in a more or less explicit manner is to build a computational graph of the function you want to differentiate and then apply reverse mode differentation to it. In fact, the backpropagation algorithm is a particular case of reverse mode differentiation. For more information, see [BPRS18].

3.1.4 Neural networks for sequence modelling

One of the disadvantages of feedforward neural networks is that the feature vectors which act as an input to the neural network must be all of the same size. However, a lot of problems can be thought as accepting a variable length sequence of elements which are not independent from each other. Problems such as speech recognition, language translation or image captioning fall under this category [ZLLS23].

The deep learning community mainly used two kinds of neural networks to tackle these kind of problems: convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The key innovation in these models is the use of parameters sharing [GBC16]. Since the variable length nature of these kind of problems prevents us from having specific weights for each feature, some weights are always used when processing the whole sequence.

From those two models, RNNs were the most successful in natural language processing tasks. However, the deep learning architecture behind all of the state of the art result is the transformer and, as a result, they are not used that much as they used to. [ZLLS23]

3.1.5 Language modelling

The task of language modelling is actually a sequence modelling task. This is because text can be seen as a sequence of tokens $x_1, x_2, \ldots, x_n$ which are fed into the model to produce the desired output. More formally, the task of a language model consists of estimating the joint probability distribution of the whole sequence [ZLLS23]:

$$P(x_1, x_2, \ldots, x_t)$$

The process of obtaining tokens from a piece of text is called tokenization and the set of

all tokens which can be generated is called the vocabulary [PH22]. The most straightforward tokenization algorithm is to consider each character as a token. The main advantages are that it is simple to implement and results in a small vocabulary. However, this representation makes it quite hard for the model to learn semantic relationships. Another approach is to use a word tokenization algorithm, in which the words seen during training form the vocabulary. Nevertheless, this solution leads to a very big vocabulary and, in addition to that, you have to deal with problem of treating out-of-vocabulary words. Moreover, if you want to use the language model for text generation purposes, the output is limited to the words seen during training. Instead, modern language models use tokenization algorithms which try to combine the advantages of those two approaches. These are called subword tokenization algorithms because a word can be splitted into several tokens. For example, a subword tokenizer may split the word *transformers* into *transform* and *ers*. During the training phase of the model, the tokenizer is first trained to learn the vocabulary from the training corpus. Byte-Pair Encoding, WordPiece or SentencePiece are examples of subwords tokenizers [MMN+24].

If we are able to calculate the joint probability distribution of a sequence, $P(x_1, x_2, \ldots, x_t)$ then we could use the conditional probability distribution, $P(x_{t+1}|x_1, x_2, \ldots x_t)$ to repeatedly sample from it and obtain text that, at least on the surface, a human could have written [ZLLS23]. There exist many sampling strategies to generate text from the probability distribution, each of which has its tradeoffs [HBD+19]. In fact, some language models are specifically trained on the task of predicting the next token given some number of previous tokens. As a consequence, they can be prompted, which means that the model uses a prompt to generate a text in a left-to-right fashion conditioned on that prompt. These are called causal language models. These models contrasts with masked language models which consider the left and right context to try to predict a token which has been masked. Given the bidirectional nature of these language models, they cannot be prompted, although some techniques exist which allow text generation. To be useful, they have to be fine tuned on the specific natural language task you are interested, such as question answering or sentiment analysis. An example of such model is BERT [MBHN24].

A term very closely related to language modelling is that of Large Language Model (LLM) [ZLLS23]. The term is usually applied to causal language models which have a lot of parameters, which are in the order of billions. These have resulted in outstanding performance on different NLP tasks. The reason why researchers are adding more and more parameters to their model is because it has been empirically shown the their performance increases as the number of parameters also increases [MMN+24].

For someone who is interacting with a LLM, these are the steps which are important. First, the prompt is splitted into tokens following the tokenization algorithm which was used to train the model. Then, a lookup table is used to obtain the embeddings which were learnt during training. An embedding is a vector representation of a token which carry semantic

information. These embeddings are then passed to the first layer of the model, which will refine that representation to take into account the context. These representation are then passed to the next layer to create new representations which contain more nuances derived from the relationship between tokens. This process is repeated across all layers in the model. Finally, the last layer outputs a probability distribution over the vocabulary, which can be used to sample from it and obtain new tokens [2].

3.1.6 Transfer learning

In the early days of deep learning, you trained a model with randomly initialized parameters[3] to solve the problem at hand by feeding it the training data. However, if we think of neural networks as models which learn different characteristic from the training data to drive the loss towards its minimum, we could reuse those learned properties for a similar task. As a consequence, the performance of those models would improve and less labeled data would be needed to train it. For example, suppose we have to create a classifier which assigns to a particular image a class from a finite set of classes. There exist many deep learning libraries which already implement a lot of classifiers so a solution would be to choose one, randomly initialize its parameters, adapt the last output layer so that it outputs a probability distribution over the classes we are considering, feed in the training data and start training. However, it has been shown that the neural networks used for computer vision tasks, which are convolutional neural networks, use their firsts layers to learn to identify borders or shapes and the latters ones are used to learn *high level* features, like a human face. This was discovered by applying different visualization procedures to the weights learnt for each layer [Mol20]. Since all of this knowledge is encoded in the model weights, it makes sense to, instead of randomly initialize the parameters, use as the initial parameters those obtained after training the classifier on another classification task. This process is called fine-tuning. Specifically, for computer vision, the most commonly used dataset to train from scratch is ImageNet [4]. Fine-tuning is the most common approach for transfer learning, but other techniques also exist [GSK+19].

Fine-tuning can also be used with models applied to NLP tasks. However, in the past, this was not the approach taken for transfer learning for these kind of problems. Instead, they used the token representations learnt by another model as the initial representations for the task specific model you choose to solve the NLP task you have at hand. The rationale behind this method is that, these representations, known as embeddings, carry some meaning so, by reusing them in another model, the model should not struggle that much compared to starting

[2] `https://www.omrimallis.com/posts/understanding-how-llm-inference-works-with-llama-cpp/`

[3] It is not random in the sense that we blindly choose the initial values for the parameters. There are some initialization techniques which take into account details of the neural network architecture, like the activation function, order to make it more likely to have a successful training process. On such initialization is Kaiming initialization.

[4] `https://www.image-net.org/index.php`

with randomly initialized weights. The main disadvantage of this approach is that you need task specific architecture [DCLT18].

The most common approach for LLMs is fine-tuning. What the companies who have the resources to carry out the process do, is to train the model on a vast corpus of text data, extracted mainly from crawling the Internet. This is the most expensive part of training a LLM and, as a consequence, this training from scratch is not done that often compared to the following phases. When the training process has finished, you have a model which is very good at predicting the next likely token. In addition to that, the model has some knowledge about human language which could be used when specializing it on a task. These are called foundation models or base models. Therefore, instead of training a model which acts as an assistant from scratch, they continue training the model on a curated question-answer pair dataset so that the model learns to follow instructions. This is the phase where companies also add guardrails to their models so that they don't answer with harmful or inappropriate content. Among the techniques used by researchers to align the answers with human preferences are Reinforcemente Learning from Human Feedback or, more recently, Direct Preference Optimization or DPO. [MMN+24]

3.2 Web development

The set of protocols and standards which allow anyone with a web browser to visit webpages is known as the World Wide Web or simply the Web. The word web is used because those webpages usually contain links or references to other webpages, making it a graph which users can traverse by clicking on them. A web application is an application which is developed using the technologies which the Web is comprised of [Hav18].

Creating a web application has become a popular option for everyone who wants to provide some kind of service to their users. Web applications are attractive for users because they only need a web browser and an Internet connection in order to use them and, as a consequence, they do not have to spend time looking for a compatible application installer for their operating system. Moreover, there exist technologies which bundle the necessary parts to run a web application (mainly a browser engine and a JavaScript runtime) with its logic so that they can be run as if they were a native application. A popular framework which achieves that is Electron [5] and a well known application developed with it is Visual Studio Code [6].

An advantage of the Web which web applications benefit from is that it is built around open standards, which allow compatibility between different web browsers. A standard is comprised of several technical documents, called specifications, which detail how some feature must be implemented. The fact that standards are open means that no single entity

[5] `https://www.electronjs.org/`

[6] `https://code.visualstudio.com/Docs/editor/whyvscode`

controls those which are approved. Instead, they are created by a group of people from different organizations and institutions who must agree on the best way to achieve the standard objective. They do that by following the motto *Don't break the Web* or, in other words, they create them in a backwards compatible way [7].

When developing a web application, there are two differentiated parts: the frontend and the backend [Ack23]. The developer who specialices in the former is known as a frontend developer while the one in charge of the tasks which belong to the latter is known as a backend developer. Sometimes, a single developer must work on the frontend as well as on the backend. These developers are referred as fullstack developers.

3.2.1 Frontend development

The frontend developer deals with the user facing elements of a web application, which mainly include the user interface. The frontend developer does not only need to know how to lay out in code the different elements which a User Interface (UI) is made of, but they should also have knowledge about designing it so that the UI does not become a barrier towards the task the user is trying to fulfill with the web application.

There exist a lot of technologies which help web developers to create web applications faster but all of them rely on three core tools, which are HypertText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript [CG23].

HTML

HTML is a document format in which the content of your webpage is written [8] [Hav18]. It defines several tags which are used to give structure to the page. For example, in order to state that a piece of text is a paragraph, you have to wrap it in a `h1` tag or, if you want to add a button, you should use a `button`. More generally, the tags are used to state the containers which you user interface is comprised of. These container may group related elements, which can be other containers.

CSS

The way these containers are styled and laid out is controlled using CSS. CSS and consists of a set of rules which apply to a set of HTML elements. The type of HTML elements it acts on depends on the selector being used in the rule. With CSS you can, for example, change the font size, margin space or border roundness as well as center an element relative to its parent HTML element or make it fit the remaining space inside of it. The CSS rules are normally written in a separate file with the extension `.css`. [9] However, a lot of people do

[7] `https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/The_web_and_web_standards`

[8] `https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started`

[9] `https://developer.mozilla.org/en-US/docs/Learn/CSS`

not like to use CSS in that way mainly because it is very easy that these file grow out of control, specially in big projects. Another reason is that some developers think that style, layout and content are so related that they should be written in the same file. Although it is possible to write inline CSS, this way of doing it does not support media queries nor pseudo-classes. Pseudo-classes allow you to apply certain CSSproperties when an HTML element is in a certain state, like focused or hovered. For that reason, there exists a project called Tailwind which offer “utility clasess". [10]

JavaScript

If you want to add some interactivity to the webpage or, more generally, execute a program in it, then you have to use JavaScript. JavaScript was created by Brendan Eich in just ten days for the Netscape browser [Sev12]. Since then, the programming language has been adopted by other browsers. Even though its name contains the name of the programming language Java, it has little to do with it.

In order to ensure that all browsers which claim to run JavaScript could actually run the same piece of code and obtain the same results, an standard is needed. That standard exists and is called ECMAScript because it was the Ecma International organization the one in charge of supervising the standardization process [Hav18].

JavaScript has been so successful that it is possible to run it outside a web browser. For example, MongoDB [11] and CouchDB [12] use it as their query and scripting language. For those who want to run it in a desktop and server environment, NodeJS [13] and, more recently Deno [14], are runtimes which can be used precisely for that.

One design decision which has significantly shaped JavaScript was making it easy for beginners to learn [Hav18]. It was thought that making the language very flexible would allow people to focus on the task they wanted to solve and they could spend more time adding functionality to their webpages instead of fighting the language. However, JavaScript was so liberal in some parts that in non trivial projects, all those features went actually against the programmer. Although some improvements have been made in subsequent versions of the language, like the introduction of a strict mode [15], other aspects like its dynamic type system make it really hard to maintain a medium-size application written in that language. For that reason, Microsoft developed TypeScript [16]. TypeScript is a programming language which adds a structural type system to JavaScript, among other features. As a consequence, the code can be statically analyzed to catch bugs before it is executed. Moreover, if the user

[10] https://tailwindcss.com/docs/utility-first
[11] https://www.mongodb.com/
[12] https://couchdb.apache.org/
[13] https://nodejs.org/en
[14] https://deno.com/
[15] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode
[16] https://www.typescriptlang.org/

defined types are given appropriate names, they can be useful when trying to understand what a piece of code is doing. However, TypeScript is not valid JavaScript because it adds additional syntax and constructs so it is necessary to convert from one to the other in a process called transpilation.

Types of web applications

The most common types of web application are server side rendered and client side rendered[CG23].

The approach which was initially employed to implement dynamic web application was Server Side Rendering (SSR). In this way of architecting a web application, the requested HTML is dynamically generated on the server and sent as a response to the client, which, in case of being a web browser, will parse it and show it to the user [17]. The reason why the HTML has to be generated when requested is because it contains data which has to be fetched from different sources, like a database, whose contents could have changed since the last time it was queried.

Initially, the component in charge of generating the HTML was not integrated in the web server but an application running on the server was responsible for that. Both entities communicated using the Common Gateway Interface (CGI), which is an interface which defines how a web server must communicate with a CGI script [18]. Even though they are referred as scripts, the programs could be written in any language as long as it understood CGI. There exist other interfaces like FastCGI [19] or uwsgi [20] which were developed to address the inefficiencies of CGI. Some popular SSR frameworks are Django [21], in which the code is written in Python or Ruby on Rails, which uses Ruby. A programming language which was specifically designed for server side scripting is PHP and it is still used in many web applications [22].

The main advantage of SSR is that the initial load time is faster than the other alternative to architect a web application because the browser does not have to do anything but to render the received HTML [23] [24]. For that reason, search engines have no problems understanding these webpages and, as a consequence, they are not penalized in their rankings. This is very important for businesses who want to appear in the firsts positions of a search result because those rankings determine precisely that. Adding the features to a webpage which make search engines to rank it higher is called Search Engine Optimization (SEO) [AMK21]. This

[17] https://hygraph.com/blog/difference-spa-ssg-ssr
[18] https://www.ionos.com/digitalguide/websites/web-development/what-is-a-cgi/
[19] https://fastcgi-archives.github.io/FastCGI_Specification.html
[20] https://uwsgi-docs.readthedocs.io/en/latest/
[21] https://clouddevs.com/django/server-side-rendering/
[22] https://w3techs.com/technologies/history_overview/programming_language/ms/y
[23] https://hygraph.com/blog/difference-spa-ssg-ssr
[24] https://www.joshwcomeau.com/react/server-components/

contrasts with the other way of making web applications, which heavily relies on JavaScript to properly work. However, SSR has its disadvantages, the main of which is that it is not suited for interactive web applications, in which the shown information frequently changes. This is because, in this paradigm, the only way to get up to date data is by requesting a full HTML from the server, which considerably impacts performance. Moreover, that limitation makes it really hard to create web applications that feel like native ones. All of that led to the development of another paradigm known as client side rendering.

In a client side rendered app, also known as a Single Page Application (SPA), the user does not download the HTML with all the application data contained there. Instead, they download a very simple HTML document whose most important part are references to JavaScript scripts. These scripts are in charge of adding the UI elements the web application is comprised of. When the user carries out some action which triggers some sort state change, the JavaScript code running on the user's browser will be in charge of modifying the UI elements appropriately (technically, JavaScript modifies the Document Object Model (DOM), which is the interface JavaScript uses to interact with an HTML document). If that change requires fetching up to date data, JavaScript will perform an HTTP request to a web server whose response will contain just the requested data, normally serialized in JavaScript Object Notation (JSON). When the data is received, JavaScript will place it on the HTML by making the appropriate modifications to the DOM [25].

```
<!DOCTYPE html>
<html>
  <body>
    <div id="root"></div>
    <script src="/static/js/bundle.js"></script>
  </body>
</html>
```

Code listing 3.1: Simple HTML document for a SPA. The `head` tag has been omitted

Even though it is possible to create an SPA using vanilla JavaScript, there should be a strong argument to choose that option over a framework. One reason for choosing a JavaScript framework is that the resulting code is less verbose than its counterpart in vanilla JavaScript. This is because in vanilla JavaScript you have to manually keep the UI in sync with the application internal state [26] and also because you interact with the DOM in a imperative manner [27]. In frameworks like React or Vue, the idea is to declaratively state how the UI should look like depending on the application state and then update that state in reaction to

[25] https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction

[26] https://daverupert.com/2024/02/ui-states/

[27] https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction

some event, like a button press or the completion of an HTTP request. These frameworks also introduce the concept of components, which group related elements and functionality of a UI so that it can be easily reused in other parts of the application[28].

One advantage of SPAs is that it is very similar to interacting with a native application because every time a link is pressed or data is requested a page reload is not needed, as opposed to the traditional SSR approach. Moreover, since the HTML is actually created in the browser, the only task of a web browser is to serve the basic HTML, the JavaScript script which will load the actual UI, CSS files and static assets like images or favicons.

However, this approach has some drawbacks. One of them is that the initial load time is not as fast as the SSR alternative. This is because, once the JavaScript has been downloaded, it needs to be executed, which takes time. On a fast computer, that is not really noticeable but on a slow one that delay may be perceived. Nevertheless, what actually make SPAs slower when loading for the first time is the extra time you have to wait until the requested data from some web service is received. In a SSR web application, when a request is received, the data source is queried and used to generate the HTML. Since the data source is normally physically closer to the machine which generates the HTML, the load time is generally faster[29]. Additionally, the people interested in SEO should be aware of fact that search engines may penalize SPAs because the web crawler, which is a software which looks for webpages to index[30], may not run JavaScript and, as a consequence may only see an HTML similar to 3.1.

3.2.2 Backend development

The backend developer focuses on the server-side part of a web application. They develop and manage the software which exposes the data and functionality which will ultimetely be consumed by the frontend.

The developer does not build everything from scratch but instead uses standard building blocks which solve known problems. For example, the problem of storing data so that it can be retrieved later is the one for which databases were specifically designed. Or the one consisting of saving the result of an expensive operation so that the next time it is requested it can be served quickly is where caches excel at. However, this does not make the process of developing backend software trivial because, for each type of building block, there exist many alternatives which were developed with a specific use case in mind. For instance, there exist many databases, some of which follow the relational model while others model their data in another way. As a consequence, it is important to know the requirements of the system being developed so that the chosen tools are well aligned with its objectives [Kle17].

[28] `https://react.dev/learn/thinking-in-react`
[29] `https://www.joshwcomeau.com/react/server-components/`
[30] `https://www.cloudflare.com/learning/bots/what-is-a-web-crawler/`

The difficulty of developing these kind of systems does not rely in just choosing the right building blocks but also in writing the stitching code which allows them to work together. All the complexity and implementations details of the system are normally hidden from clients, who only interact through an Application Programming Interface (API).

There exist many ways to allow clients to interact with the services provided by a system. For example, you can create a custom protocol at the transport layer to achieve exactly that. However, it is better to rely on protocols which are well known because there exist a lot of tools which understand them and, as a consequence, allow you to focus on higher level details rather than on lower level ones, like the wire format of the data. The most common protocol used for the development of these kind of APIs is HyperText Transfer Protocol (HTTP).

When designing these kind of systems, one should have the objectives of reliability, scalability and maintainability in mind. It should be noted that there does not exist explicit rules which tell you how to develop systems which satisfy those properties but some patterns can be followed to achieve that task [Kle17].

Reliability

A system is reliable when it continues working in spite of things going wrong [Els05]. A concept which is closely related to this one is that of a fault-tolerant system. A fault and a failure are not equivalent. The former refers to the situation of the system doing something which the requirements did not state (it deviates from its spec) while the latter is used when the system completely stops offering its services to clients. If not handled correctly, faults may lead to failure.

People tend to think that, when a system is fault-tolerant, then it is impossible for the system to fail. However, the probability of failure cannot be reduced to zero because you cannot prepare it for absolutely everything but, with the right tools and procedures you can get close to it. A system has three sources of faults: hardware faults, software bugs and human errors (i.e misconfigurations which leads to an outage).

Scalability

The concept of scalability describes the ability of a system of still offering its services in spite of an increase in load [Kle17]. In order to measure this property of a system, you need to first define some load parameters, which depend on the system. An example of a load parameter could be the number of requests received per second or the number of players simultaneously playing a game. When you know the load of the system, you should ask yourself how the system is affected when a load parameter increases and the resources you have to add to it to cope with it. The answers, which require you to take some metrics, will tell you how scalable the system is with respect to the load parameters under consideration.

Therefore, it does not make sense to say that the system is scalable or does not scale well

because it depends on the load metric you are considering. It is possible the system can cope with an increase in load in some part but completely stop offering its services when the pressure increases in another part.

Maintainability

The maintainability of a system is a measure of how easy it is to apply modifications to it [VRW+16]. Some examples of those modifications include adding new features, fixing bugs or adapting it to new platforms. In order to ensure the maintainability of a software system, it is recommended to follow the principles of good operability, simplicity and evolvability.

By good operability we refer to making routine tasks, like monitoring or keeping dependencies up to date easy to perform. It is called operability because the operations team is in charge of that. Moreover, by keeping the system simple, it easier to understand and the probability of introducing bugs when applying modifications is reduced. This is because simplicity makes it easier to reason about the system and it is clearer the consequences the introduced modifications can cause. Finally, the evolvability principle is very important nowadays because it is unlikely the system requirements will remain fixed. As such, you need to create a system which is open for extension to quickly adapt to those changes. In fact, Agile software development methodologies were specifically created to allow developer teams to deliver the changes at the requested pace [GGI20].

3.3 Extended reality

Extended Reality (XR) is an umbrella term which covers the concepts of Virtual Reality (VR), Mixed Reality (MR) and Augmented Reality (AR) [PMB22]. All of them have as a key characteristic the ability of immersing the user into an environment in such a way they feel they are there. The main difference between them lies in the amount of real world information they are incorporating into it, being VR the one which does not include any elements and AR characterized by the opposite. MR is located more or less at the middle of that spectrum .

XR is a key technology in what is known as the metaverse. The term appeared for the first time in a novel by Neal Stephenson named *Snow Crash*. It was envisioned as another world which users could access by using the appropriate XR devices. Currently, with the help of other technologies like 5G, blockchain and AI, the existence of that parallel world is being close to become a reality. Even though the metaverse is a recent concept and its definition slightly changes as technology advances, big companies like Microsoft, Tencent, and NVIDIA are starting to show interest in it. This is specially true in the case of Meta, which was previously called Facebook but decided to change it to reflect their bet on the metaverse [WSZ+22].

3.3.1 XR Hardware

In this section, the different kinds of hardware there exist to achieve the type of experiences and interactions which characterize the XR are overviewed.

Head Mounted Display (HMD)

In order to achieve an immersive experience, it is not only necessary to view the virtual environment but also to freely navigate through it. In order to achieve that, HMDs were designed. Ideally, the user should not wear any device to enter this virtual environment but unless scientists come up with a way of implementing these kind of systems in space, the easiest solution is to use those head-worn devices [RH05].

An HMD is a headset that contains one or two screens (one for each eye) which cover the persons's eyes. These are used to show computer generated pictures or images captured from the real world using cameras. An HMD is usually equipped with different kinds of sensors which allow the user to move around in the virtual environment as they would do in the real world. Additionally, it is possible to add other elements which increase the interactivity such as speech recognition systems or gloves [SC03].

(a) Meta Quest 3 (October 2023) (b) Pico G3 (May 2023)

Figure 3.2: Two modern HMDs
The Meta Quest 3 supports AR thanks to its front cameras but the Pico G3 is a headset exclusively for VR

However, designing an HMD which users can use comfortably is challenging. This is because the display specifications these kinds of devices must met go against each other [RH05]. As a consequence of that, health related issues such as dizziness, nausea, vomiting and cold sweats have been reported. One of the most well known problems with **HDM!** is motion sickness, which may cause the aforementioned health problems. In fact, it is estimated that one in third users meet the necessary requirements to suffer from this condition. As a result, this severely limits the practical application of XR using HMDs for a significant number of people [CJA+20].

The devices which allow people to immerse in virtual environments have not always been

available to the general public. There was a time in which its use outside simulators tailored to surgeons, military and pilots was rare. However, that changed in 2013 when the company Oculus released a series of headsets affordable for consumer budgets. This encouraged other companies to create their own hardware for XR. These new HMDs were not only cheaper but also incorporated technical advancements such as the increase in the Field Of View (FOV) range [JK18].

Currently, the headsets available for sale can be consulted on the webpage VRcompare[31]. The page breaks down different technical details for each of the headset but one aspects which is shown immediately after you click to see more details about a specific HMD is whether it is standalone or it is actually PC powered. The difference lies on the fact that standalone headsets are self-contained pieces of hardware which contain everything needed to run an XR application while PC powered headsets require a powerful PC in which expensive calculations are offloaded [32]. The main advantage of standalone headset is that they are portable since you do not have to worry about having a PC close to you and, as a consequence, they are more convenient for some practical applications. However, the graphics quality is not as high as the one provided by PC powered headset. Moreover, these latter ones are generally considered to be more robust and accurate in movement tracking at the cost of being harder to setup and not being portable.

A point which developers interested in creating applications that incorporate elements from the real world have to pay attention to is the headset support for pass-through. This feature allows the user to see the real world without having to take off the device. Not all headsets have support for pass-through, like the Pico G3 [33]. Others, like the Oculus Quest 2 have a partial support for it by leveraging the hand tracking cameras. However, the captured images are in greyscale [34]. Nevertheless, there exist alternatives which have been designed with AR applications in mind and, as such, offer full color pass-through. Some examples include the Meta Quest 3 [35] or the Apple Vision Pro [36]. The fact that there exist HMDs which support full color pass-through does not mean that developers creating applications for those headsets have access to the data captured by the cameras in real time. This is the case of the Meta Quest 3, which blocks its access for "privacy reasons".

Mobile augmented reality

Even though HMDs are the main way to interact with the possibilities which XR offers, other approaches exist. Specifically for AR, mobile augmented reality is a very promising

[31] https://vr-compare.com/

[32] https://www.vr-wave.store/blogs/virtual-reality-prescription-lenses/which-headset-should-you-buy-pcvr-vs-standalone-vs-console

[33] https://vr-compare.com/headset/picog3

[34] https://vr-compare.com/headset/oculusquest2

[35] https://vr-compare.com/headset/metaquest3

[36] https://vr-compare.com/headset/applevisionpro

alternative since it makes use of hardware which you can easily take with you wherever you go. Some examples of mobile devices in which AR applications can be run include smartphones and tablets. While this approach has its advantages, like its low cost compared to other alternatives or the fact that almost everyone can easily experience AR because those devices are nowadays ubiquitous, other set of challenges arise such us the limited resources that smartphones and tablets offer to AR developers compared to special purpose hardware like HMDs or the environment where the user is located when they are executing the AR application, which may lead to an undesired level of immersiveness [Cra13].

AR glasses

For AR solutions, there exist another kind of device on the market which tries to be more lightweight and comfortable than HMDs. This is the case of Augmented Reality Smart Glasses, which are AR devices that are "worn like regular glasses and merge virtual information with physical information in a user's view field" [RBR15]. The specialized web VRcompare also gathers information about different kinds of AR glasses. One of the most popular AR glasses according to that webpage are the Xreal Air 2 Pro [37] which were released in 2023 and costs $410. However, they are not standalone because they require a smartphone to work. A pair of AR glasses which do not need an external device in order to use it are the Magic Leap 2, released in 2022 [38]. However, with its price of $3299, they are not affordable by a typical consumer budget.

(a) Xreal Air 2 Pro (November 2023)

(b) Magic Leap 2 (September 2022)

3.3.2 Developing XR applications

Since computer generated images are a key aspect in XR applications, the most appropriate software to develop them is a real-time 3D engine [AMHH19]. By real-time, we are referring to images which are rendered almost immediately and, as a consequence, it is possible to show a series of them with such a small delay that the result of some kind of interaction, like a button press, is quickly noticeable on the screen. The most common use of a real-time 3D engine is game development, but this kind of software incorporates additional tools to

[37] https://vr-compare.com/headset/xrealair2pro
[38] https://vr-compare.com/headset/magicleap2

develop other kinds of applications, such as simulations. [39]. Unity [40] and Unreal Engine [41] are examples of real-time 3D engines.

One of the main challenges real-time 3D engine vendors have to face is the support for a significant number of XR devices. As it usually happens with a new piece of technology, each company in the XR hardware market designed its own protocols to interact with their products. As a consequence, this severely limits the number of devices an XR application can target. Unity tried to solve this problem using a plugin system which allows Unity applications to use a common interface to communicate with all supported devices. The vendors only have to implement the interfaces that the plugin system requires [42]. However, this is a good solution if the developer wants to create their application with Unity. If a very promising game engine like Godot [43] wants to add support to develop VR games, their developers can opt for the aforementioned plugin system solution to solve the support problem, but XR device vendors would have to implement another interface.

To avoid those complications vendors and developers could agree on a standard. For XR, that standard exists and is called OpenXR [44], which is maintained by Khronos group[45]. OpenXR is open and royalty-free. This means that anyone can propose changes to it and that its use is not restricted by licenses fees. From the point of view of the programmer, XR is is just a set of APIs which they use to, for example, access a controller state or submit rendered frames. For implementors, OpenXR tells them the functions which controls the XR device as well as establishing the lifecycle of a XR application. In addition to that, OpenXR has been designed to be extensible so that newly added features to a device can be supported without resorting to vendor specific solutions. In fact, companies can submit their extensions so that they become part of the standard [46].

These APIs are fairly low level so, if you are developing with a real-time 3D engine, the most common thing is to work with a friendlier API built on top of it provided by the engine. For the specific case of Unity, since they already have a plugin system, they created a plugin which makes all OpenXR compliant devices a target for a XR application developed with it.

[39] https://learn.unity.com/tutorial/what-can-unity-do
[40] https://unity.com/
[41] https://www.unrealengine.com/en-US
[42] https://docs.unity3d.com/Manual/XRPluginArchitecture.html
[43] https://godotengine.org/
[44] https://www.khronos.org/api/index_2017/openxr
[45] https://www.khronos.org/
[46] https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html

Chapter 4

Methodology

In this chapter, the methodology which has been followed is explained as well as the reasons for choosing it over other options. Then, the different working packages in which the project has been divided are detailed. The chapter finishes by detailing the different hardware and software resources which have been employed over the course of the project development.

4.1 Development methodology

A software development methodology normally falls under one of two categories. On of them is the waterfall approach [Jon17], which was the way in which initially software was developed. Even though for well-defined project a waterfall approach may be a good option, in settings were continuous feedback should be given to ensure the developed software is aligned with the desired objectives or the requirements are constantly changing, the model starts to fall apart. Since this project makes use of tools which are relatively new which I do not know in advance whether the will work as planned, the rigid planning which characterizes the waterfall model can only slow me down.

Agile methodologies emerged as an alternative approach to organize the development of software projects which were not as rigid as the waterfall model [DNBM12]. Although there exist many Agile methodologies, all of them follow the principles outlined in the Agile manifesto [1]. Popular methodologies which are grounded on those principles include Scrum, Kanban, and Extreme Programming (XP).

Even though the aforementioned popular methodologies are considered more lightweight than the traditional way of developing software, I, nevertheless, consider that they still introduce too much overhead in solo developer projects, as this one. For that reason, I opted for a simpler one which still followed the agile principles. The chosen one has been Adaptive Software Development (ADS).

[1] https://agilemanifesto.org/

4.1.1 Adaptive Software Development

In a software project which involve technologies like LLMs or XR devices, the results are unpredictable. As a consequence, the determinism which characterizes the act of planning is not useful in these kinds of settings. ADS proposes the Speculate-Collaborate-Learn cycle to quickly adapt to the unexpected circumstances which arise as a consequence of that unpredictability [Hig13].

- **Speculate**. Since planning is associated with a deterministic output given some inputs, that word is instead replaced by speculate. This word admits that we could be wrong and that the best strategy is to remediate it is to put the mechanism to adapt.
- **Collaborate**. In this phase, the predictable parts are identified and the work environment is accommodated so that the emergence of the final product can happen. For this project, these guidelines have been followed every time I had to work on the tasks associated to a work package.
- **Learn**. In this part of the cycle, the product is shown to the stakeholders to receive feedback from them so that the next iteration of it is closer to what they want. This usually leads to a change in the requirements which which brings us to the speculate phase again.

4.1.2 Work distribution

Given that there does not exist a fixed planning, the project was divided into several work units which were added after feedback was received from my advisors. These work packages are detailed below. The period of time I spent on each one is shown in the Gantt diagram of figure 4.1.

1. **Research about the tools used for the development of web applications as well as those used for creating XR applications**. Given my lack of experience developing web applications and software for the Meta Quest 3, some time had to be spent researching about them.
2. **Development of the protocol editor**. After analyzing the different ways in which protocols could be shown, the frontend for the protocol editor was developed. This work package also includes the development of the page which allows the user to edit, delete or create one.
3. **Implement backend logic which handles the protocols**. This work package includes the set of tasks related to designing and implementing the part of the protocol service in charge of managing the protocols.
4. **Implement protocol visualizer on the Meta Quest 3**. A visualizer to show a protocol stored in the backend was designed and implemented. This work package also includes the protocol menu which allow the user to choose one.

5. **Research about including LLMs to the system and the possibility of self-hosting one**. Given the hard work people have to put into developing the steps which comprise a protocol from documentation, the possibility of using a LLM which acted as an assistant during the protocol execution was considered. The possibility of self- hosting one was also analyzed.
6. **Implement LLM service**. After selecting the proper technologies, the LLM service was created to interact with the LLM.
7. **Interact with LLM service using the web application**. Support to interact with the LLM from the web application was added as well as the ability to upload documents containing relevant information during the execution of a protocol.
8. **Interact with the LLM service from the XR application**. They way in which the Meta Quest 3 user interacts with the LLM was envisioned and implemented.
9. **Application deployment**. Each component of the system was containerized and the appropriate software was installed and configured on each server.

A Gantt chart with the aforementioned work packages is shown in figure 4.1

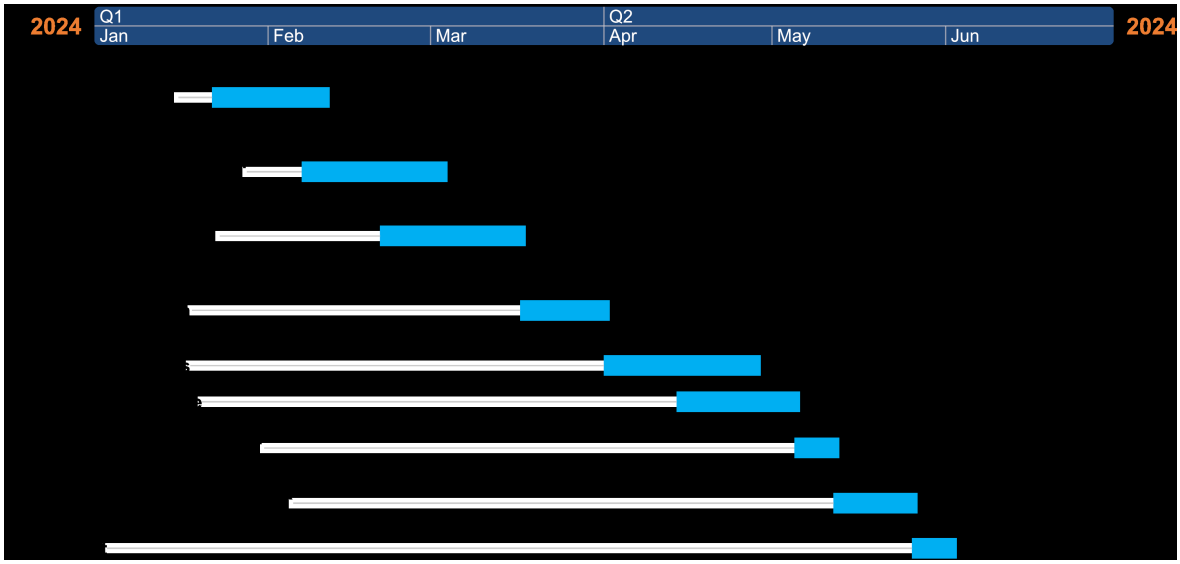

Figure 4.1: Gantt chart of the work packages

4.2 Development workflow

Git[2] is the version control system which has been employed for the development of the project. The source code is hosted on Github[3]. The commit messages follow the Conventional Commits specification[4]. By following it, the intention of the commit is clearer because the ways to express the reasons to perform the change are standardized in the specification.

[2] https://git-scm.com/
[3] https://github.com
[4] https://www.conventionalcommits.org/en/v1.0.0/

Moreover, since it also describes the format the commit messages which adhere the specification must follow, they can be processed by automatic tools.

In addition to that, when a new feature had to be implemented, a new branch was created for it and a pull request was open. The description of the pull request was filled out with the scope of the feature. The messages of the commits found in those feature branches do not follow the Conventional Commits specification and, in fact, they are usually not very descriptive. This is because those commits are necessary for the gradual development of the feature but I do not consider them important enough to be included in the commit history of the project. To avoid including them, the squash merge strategy has been utilized. This strategy merges the commit of one branch into another by creating a single commit with all the changes which have been made with respect to the target branch. As a consequence, if the commit history is consulted, it looks like as if the feature was implemented in a single commit. However, when a branch backed by a pull request is merged with this strategy on Github, a link to it is added to the commit message. In this way, if more context about the decisions which lead to the implementation is needed, you click on it to see the pull request in which it was developed.

4.3 Hardware and software resources

In this section, the different kinds of hardware and software resources which have been needed to carry out the project are detailed.

4.3.1 Hardware resources

- **Meta Quest 3**. This is headset for which the XR application was developed. It is suited for AR applications since it supports full color pass-through. They were released by Meta in October 2023. The headset that I have used for the development of this project was lent by the UCLM spinoff Furious Koalas.
- **My Personal Computer (PC)**. This is the computer I normally use to work. I had to install an additional disk because Unity projects tend to be heavy and I did not have enough disk space for them. It includes a Intel Core i7-6700 CPU, a NVIDIA GeForce GTX 1060 6GB GPU and 16 GB of RAM.
- **ITSI computer**. This computer has been used for testing the LLM service until the AIR Research Group received its server. It is located in the *Instituto de Tecnologías y Sistemas de la Información* building. The computer is owned by UCLM spinoff Furious Koalas.
- **AIR Research Group server (*airproy*)**. This was bought by the AIR Research group to run compute intensive workloads. For that reason, this is where the service used to interact with the LLM is running. It includes a 13th Gen Intel Core i9-13900K CPU, a NVIDIA RTX 4000 ADA 20GB GPU and 126 GB of RAM. This machine is also

referred as *airproy*.

- **IONOS Virtual Private Server (VPS)**. A VPS is a server in which the hardware resources have been virtualized. As a consequence, it can be used by different users which are isolated from each other. The VPS provider is the company IONOS and the chosen one from the available ones [5] was VPS Linux M.

4.3.2 Operating systems

- **Fedora 39**. Fedora is a Linux distribution sponsored by Red Hat and, in fact, it is the upstream of RHEL [6]. It is developed by the Fedora Project and it differentiates itself from other distributions by including the latest technologies without compromising very much the stability of the system. This has been the Operating System (OS) used for developing the frontend and the backend and it is installed on my PC.

- **Fedora 39 Server**. It is a version of Fedora which only contains useful software for administering a server. This was installed on the ITSI computer. This OS was chosen over Ubuntu Server 22.04 because its installer crashed in the middle of the installation process for unknown reasons.

- **Ubuntu Server 22.04**. It is a version of Ubuntu 22.04 which only contains useful software for administering a server. It powers the AIR Research group server as well as my VPS.

- **Windows 10**. Windows was needed to develop the application for the Quest 3 with Unity. Even though Unity is multi-platform, a lot of problems are avoided by choosing Windows. The same happens with the needed software to connect the Quest 3 to a PC. It is installed on my PC on a separate disk.

4.3.3 Software resources

For the development of the system, different software tools have been employed. The tools have been categorized into four distinct classes based on the general problem they were designed to solve: programming languages, development tools, software libraries and documentation tools.

Programming languages

- **C#**. It is the scripting language used by Unity Engine. The language was developed by Microsoft. Unity allows you to choose how the C# code is transformed when building the application by using different scripting backends. For this project, the scripting backend IL2CPP was chosen.

[5] `https://www.ionos.es/servidores/vps`
[6] `https://docs.fedoraproject.org/es/quick-docs/fedora-and-red-hat-enterprise-linux`

- **Python**. Python is a programming language which is characterized by its ease of use. In the last years, it has received a lot of attention thanks to the software libraries and frameworks developed to work with machine learning models but it is also a good choice other the kinds of applications. It has been the chosen language for the backend.
- **TypeScript**. TypeScript is a programming language developed by Microsoft. Its main objective is to add a structural type system to JavaScript, among other features, so that the written code is more maintainable and some kinds of bugs are caught earlier. It has been used to develop the frontend.

Development tools

- **Unity Engine**. It is a 3D real-time engine which is well known for its use for game development. It also allows you to create XR applications and, as a consequence, has been chosen for developing the XR application.
- **NGINX**. It is a web server which can also act as a reverse proxy. It has been used for serving static files but its reverse proxy capabilites have made it an API gateway [7] to the services which constitute the backend.
- **MariaDB**. It is a Relational Database Management System (RDBMS). It is a fork of MySQL. It has been used to store all the data which needed to be persisted except for static files and the chunks in which documents are divided.
- **Qdrant**. It is a vector database [HLW23]. These kind of databases are specifically designed to store vector embeddings in such a way that retrieving the ones which are the most similar to a given vector according to a similarity measure is done efficiently. It has been used to store the chunks in which user uploaded documents are partitioned.
- **Visual Studio Code**. It is a multiplatform source code editor developed by Microsoft. It is characterized by its ease of use and its support for many programming languages by means of the installation of the appropriate plugins. This code editor has been the one used to write Python and TypeScript.
- **Visual Studio**. It is an Integrated Development Environment (IDE) developed by Microsoft exclusively for Windows systems. It is mainly targetted for C# and C++ developers. It has been used to write the C# scripts for the Unity application.
- **Git**. It is version control system originally developed by Linus Torvalds. It is used to track the changes made to the files which are part of a software project and supports different kinds of workflows which allow programmers to remotely collaborate on the same codebase.
- **Github**. It is a Git repository hosting service which also includes features which facilitate the collaboration between software developers, such as Pull Requests. The whole

[7] `https://www.redhat.com/en/topics/api/what-does-an-api-gateway-do`

project is in a single repository comprised of different directories which contain the different parts of the system.

- **Docker**. Docker is a software which uses Linux kernel features, like namespaces and cgroups, to provide virtualization at the OS level in the form software packages known as containers. These containers have all the necessary dependencies to run a single application and, as a consequence, a lot of problems which arise when running an application on an environment different from the one used to develop it disappear. It is used in the development stage of a software project but it can also be used in production.

Software libraries and frameworks

- **NextJS**. It is a React framework developed by Vercel. It adds additional features needed when developing a web applications, such as a routing system or image optimization, which React lacks of. It also allows you to use React Server Components. This framework has been used to develop the frontend.
- **ReactFlow**. It is a React library which aims in the development of node-based UI. It has been used to implement the protocol editor.
- **Zustand**. It is a state management library for React. It has been needed to manage the protocol editor state.
- **MUI**. It is a React component libraries which offer different kinds of UI elements which adhere to the Material Design style guide.
- **llama.cpp**. It is a Command Line Interface (CLI) program and library written in C/C++ to run LLMs locally. It supports more models apart from the ones from the LlaMa family and the list of available ones is still growing.
- **python-llama-cpp**. These are the Python bindings for llama.cpp.
- **LlamaIndex**. It is a Python library which contains tools which ease the development of RAG (see section 5.4.4) applications.
- **FastAPI**. It is a Python web framework for the development JSON APIs. It keeps the code simple and readable by making use of Python type hints.
- **SQLAlchemy**. It is one of the most popular Object Relational Mapping (ORM) for Python. An ORM reduces the boilerplate code that results from the *impedance mismatch* that exists between the relational model and object oriented paradigm [Kle17].

Documentation tools

- **Overleaf**. It is an online LaTeX editor which enables different people to collaborate on the same document. It has been used to write this technical report and receive feedback from my advisors about the written text.
- **excalidraw.com**. It is a web application to create different kinds of diagrams. It has been used to draw the diagram of own creation.

Chapter 5

Architecture

THIS chapter is devoted to explaining the different parts which constitute the system as well as showing how each one interacts with the others. Firstly, we will give an overview of the system as a whole. Later, the design of each subsystem will be detailed with an emphasis on the most relevant techniques and the trade-offs each important decision has had.

5.1 Overview

The architecture that underpins the system is comprised of three differentiated parts.

i) **Web application**. It is what the user employs to design their protocols, associate resources such as images to each step and attach documents containing useful information which may be leveraged by a LLM to produce answers based on the knowledge contained there.

   In the context of this work, a protocol is defined as procedure consisting of several steps. These steps contain information about the actions the user has to carry out. Each step contains one or more options from which the user chooses one and that decision determines the next step during the execution of the protocol. This dynamic is repeated until a final step is reached. A lot of medical procedures, such as some diagnoses, can be represented in that way, specially if they are in the form of a flowchart.

ii) **XR application**. This is the application which has been developed for the Meta Quest 3 headset. This application makes use of the capabilities that MR offers to present the information introduced by the user using the web application in novel ways. Both applications are actually frontends which expose, in a friendly manner, the functionality and data provided by the backend

iii) **Backend**. The main tasks of the backend are to persistently store the data needed by the system to work, enforce some constraints on it according to the requirements which define what is considered valid data, and interact with a self- hosted LLM.

Although the backend looks like a single piece of software to the user, from the point of view of the system designer, it is actually a distributed system composed of two parts. One

of them is a HyperText Transfer Protocol (HTTP) server serving a JSON API running on a VPS which interacts with a database and saves user uploaded resources, such as images and documents. The other one is another JSON API which is in charge of querying a self-hosted LLM and processing and storing the information contained in the user uploaded documents in a format efficient for retrieval and appropriate to be used by the model. This service is being executed on a server which has the necessary resources that these kinds of models demand. For a thorough discussion about the reasons which lead me to split the backend in such a way, see section 5.6.

A diagram of the whole architecture containing the different components of the system with its most important functionalities and elements is shown in figure 5.1.

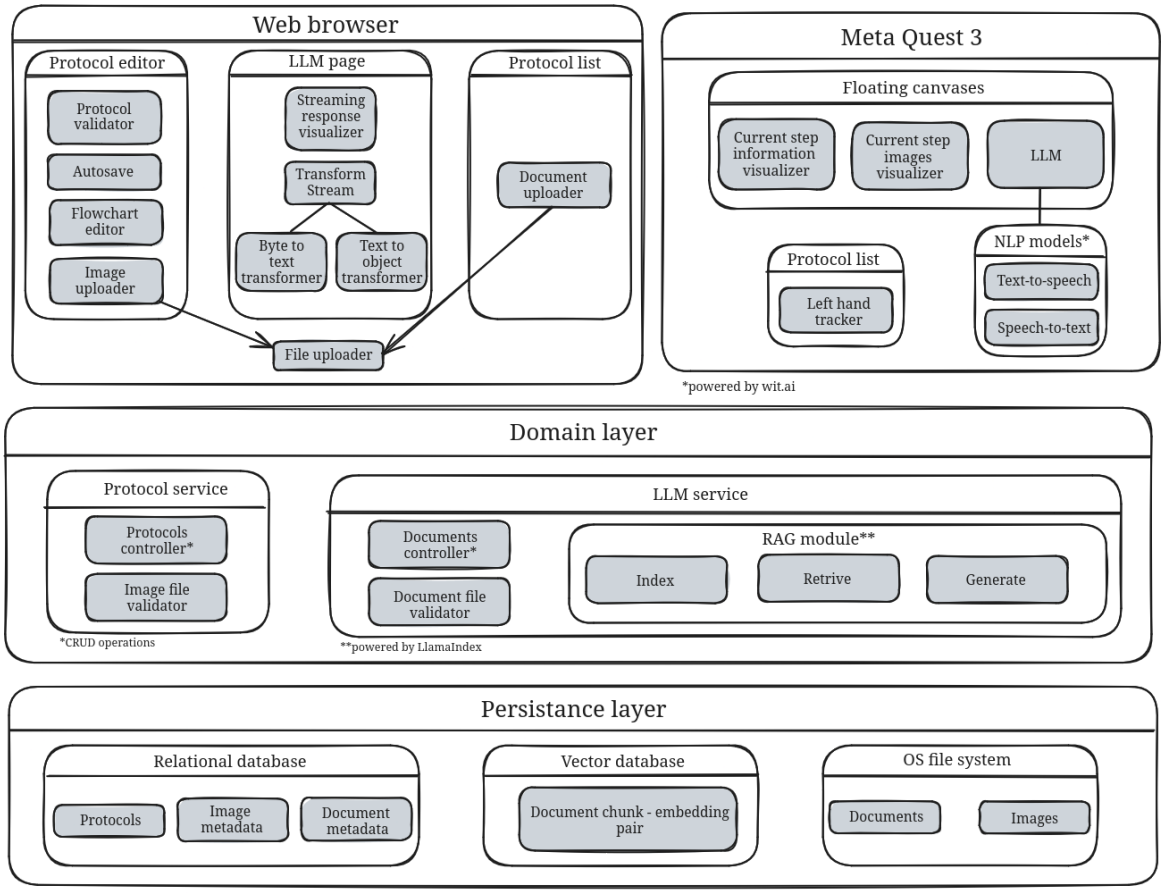

Figure 5.1: Architecture of the system

5.2 Web application

One of the first decisions which had to be made regarding the web application is whether to use a Server Side Rendering (SSR) approach or instead build a Single Page Application (SPA) [CG23]. I opted for the SPA solution for several reasons. Firstly, SSR is a good option for applications whose state can be handled on the server. A subset of those ones include Create Read Update Delete (CRUD) applications, which mainly consists in querying

a database and returning the results or modifying its contents using the data inserted by the user in some kind of form.

However, two features of the SSR approach forced me to completely discard it. On one hand, an SSR application returns HTML. If the device which consumes the data is a web browser, there is no problem because the data is already in the format that the web browser employs to show it. However, in my system there is another device which has to have access to the same data but does not understand HTML out of the box. As a consequence, a representation for the data decoupled from the format used by the device to show it is needed. One of the most commonly used nowadays is JavaScript Object Notation (JSON), which is the one I have chosen. Therefore, the web application has to consume JSON and that it is something which a SPA, possibly with the help of a framework or library, allows you to easily perform.

The other reason for choosing a SPA over a SSR application is that the web application includes a flowchart editor whose state is constantly changing when you use it. For example, a node has to update its position when you drag it or its border has to change its color when you select it. It would be extremely inefficient to keep that state only on the server because every time it needs to be updated a network request has to be made to notify it. It is better to handle all of that client side and sync the local state with the server periodically. For more information on how that syncing has been done for this application, the reader is encouraged to refer to subsection 5.2.1. These kinds of applications, in which there exist a lot of interactivity which causes many User Interface (UI) elements to change in reaction to a state update, are the ones for which SPA frameworks like React [1], Vue.js [2] or Svelte [3] were designed for.

NextJS is the framework in which the web application frontend has been developed. NextJS is a React framework. This means that it provides tools which solve problems which frequently appear when developing web applications which React does not offer by default, like routing [4]. It also includes something called React Server Components (RSC), which allow you to render some part of the HTML on the server while those that require client-side state are rendered on the browser using JavaScript.

The web application has four views. These can be accessed by clicking on the buttons located on the left sidebar:

- **Welcome page**. It is the one which appears when you type the root Uniform Resource Locator (URL). An explanation of this project and other useful information can be found there.

[1] `https://react.dev/`
[2] `https://vuejs.org/`
[3] `https://svelte.dev/`
[4] `https://react.dev/learn/start-a-new-react-project`

- **Protocol list page**. This view contains the list of the protocols the user has created. Each protocol is represented by a card in which the protocol's name and two buttons to edit it or update it are found. In the upper right corner of the card, there is a button that, when pressed, displays a menu with two options. One option opens a dialog which is used to upload the documents containing the information the LLM incorporates into its answer when it is queried and the other takes you to the LLM page. This page is shown in figure 5.2.
- **LLM page**. This is the page which allows the user to interact with the self-hosted LLM. There exist two versions of this one. One uses the information contained in the documents associated to a protocol but, in the other, it does not make use of any external knowledge. The former version is accessed by clicking on *LLM* button located on the sidebar while, for the latter, you have to click on one of the options in the menu displayed when you press the button located in the upper left corner of a protocol card.
- **Protocol editor page**. This is where the user creates their protocols. It also makes it possible to attach images to a step. The dialog which enables the user to do that was based on the one designed to upload documents. This fact is represented in the global architecture diagram of figure 5.1.

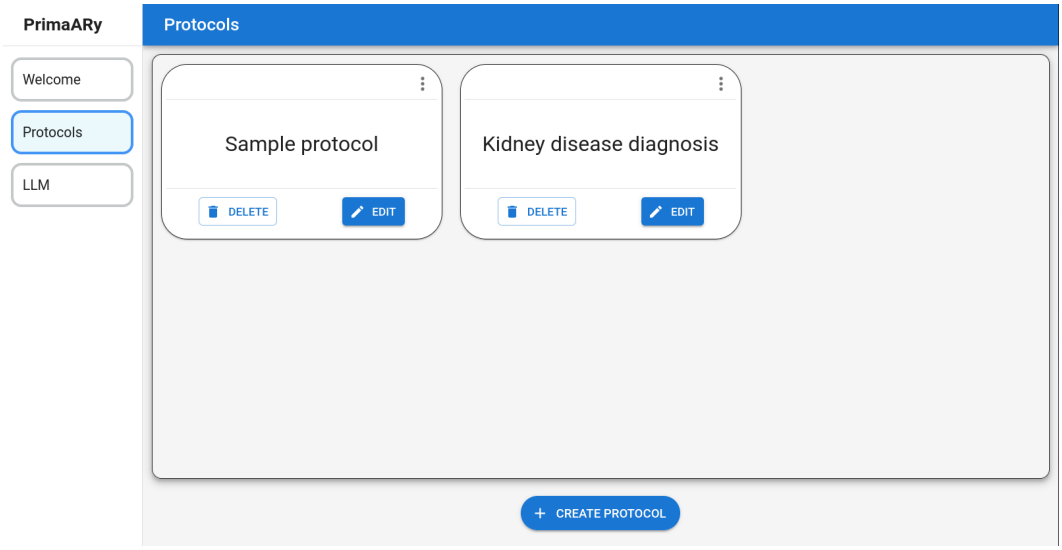

Figure 5.2: Protocol editor page

From the aforementioned pages, the most challenging ones have been the protocol editor and the LLM view and, for that reason, both are detailed in the following sections.

5.2.1 Protocol editor

In order to create or modify a protocol, a simplified flowchart editor has been developed. However, other options were considered to solve that problem. The most appealing one consisted in developing a custom Domain Specific Language (DSL) which the user could use

to formally state all the elements needed to successfully carry out a protocol. However, I did not find this solution satisfying because of the target audience of this application. I expect this application to be used by people who probably do not have any experience working with a DSL. For that reason, they would have to spend a period of time learning the DSL before having a working protocol. Similarly, if the user does not frequently design protocols with the DSL, it is very likely that they forget how to create one after some time of not using it. As a consequence, they would have to relearn the DSL, turning it into an important obstacle towards the real objective the user using this application has, which is defining a protocol. Other approaches were explored but all of them resulted complex UIs which hurt usability.

The idea of employing a simplified flowchart editor came up after realizing that these tools have actually been used in medical contexts to specify some kind of procedure. An example can be seen in figure 5.3, which was uploaded by the National Institutes of Health (NIH) to its webpage [5]. Flowcharts are also a good tool to summarize the information obtained for the development of an expert system since they simplify the process of obtaining the rules from it [Rod15].

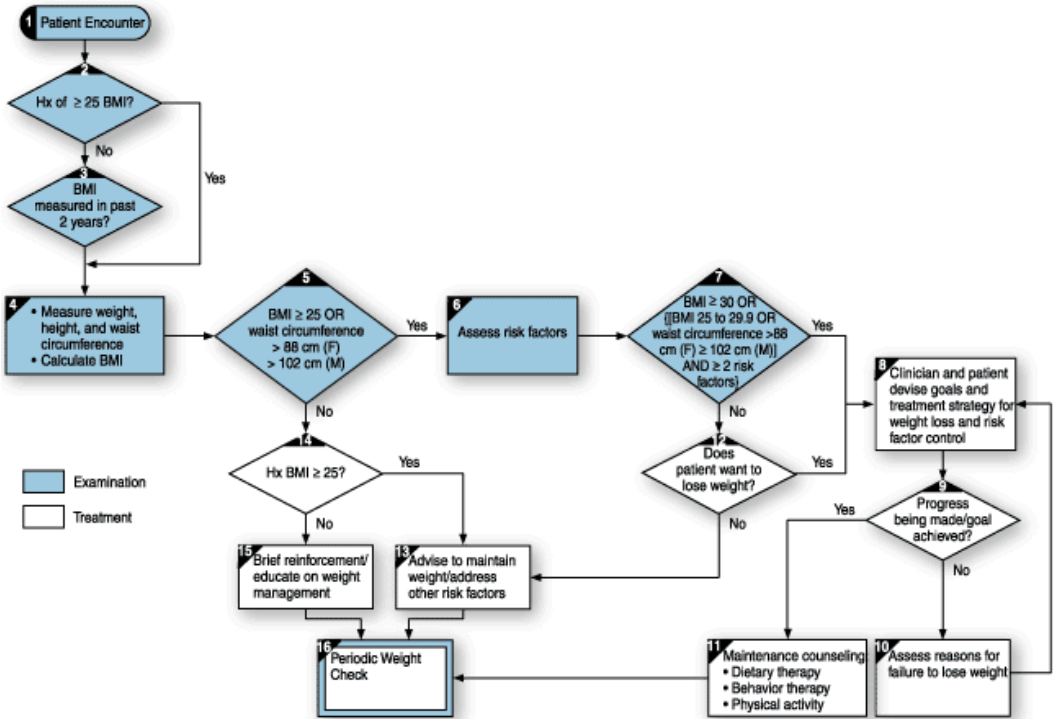

Figure 5.3: Example of a real medical procedure represented using a flowchart

The reason why I call it simplified flowchart editor and not just flowchart editor is because it does not make use of the symbols which characterize these kind of diagrams. There

[5] https://web.archive.org/web/20170502044017/https://www.nhlbi.nih.gov/health-pro/guidelines/current/obesity-guidelines/e_textbook/txgd/algorthm/algorthm.htm

even exist standards which state how flowcharts should look like [6]. However, this diagram represent a flow of different steps so I think it is appropriate to keep the word flowchart.

The flowchart editor

The flowchart has been developed with the help of a React library called ReactFlow [7]. Before discovering that library, I thought on using p5.js[8], a JavaScript library which provides an easier API to create graphics and elements with the canvas HTML element. Had I chosen p5.js , it would have taken more time to develop the flowchart because I would have needed to code every behaviour from scratch apart from integrating that code with the rest of the React application.

When you create a new protocol, there is already one node which represents the initial step. To create another protocol, you have to click on one of the four handles a node has and drag the mouse to the position where you want to place the new node. When you release the button, the node is automatically created at that position. To modify the name or description of the new node, you have to double click on it. That will open a menu on the right part. From that menu, you can also attach images to the protocol step. Deletion of a node or an edge is achieved by selecting it and pressing backspace. Finally, to modify the label of an edge, you must double click on it.

Choosing the tool to build the editor

Since React Flow has been specifically designed to create node-based UIs, it already incorporates basic behaviours so the developer can focus on the application logic. Specifically, React Flow handles the updating of the nodes position when you perform a pan gesture or when you drag one or several nodes. It also frees you from having to implement the logic which updates the UI when you zoom in or zoom out. In addition to that, it provides an easy mechanism to design custom nodes and already integrates the functionality which allows nodes to be connected by edges, which can be customized and may contain extra information like a label. Finally, it allows the user to delete nodes and edges by pressing the backspace key after selecting those which they want to be deleted.

For the most important high level events, such as deleting or dragging nodes, React Flow allows you to pass a callback which will be called by the library every time it detects it happens so the task of designing the UI with React Flow mostly reduces to identifying those event that trigger some change in the editor state and react appropriately them.

Internally, the flowchart is stored as a graph. The graph is represented by two arrays: one contains the nodes and the other the edges. Each node has an unique id, which is generated

[6] https://cdn.standards.iteh.ai/samples/11955/1b7dd254a2a54fd7a89d616dc0570e18/ISO-5807-1985.pdf

[7] https://reactflow.dev/

[8] https://p5js.org

using the library `nanoid` [9]. An edge references its source and target node using that id. I decided to use that representation instead of other because that is the one React Flow employs. If I had used a different one, I would have had to convert my representation to the one React Flow accepts. The described graph representation is shown in figure 5.4

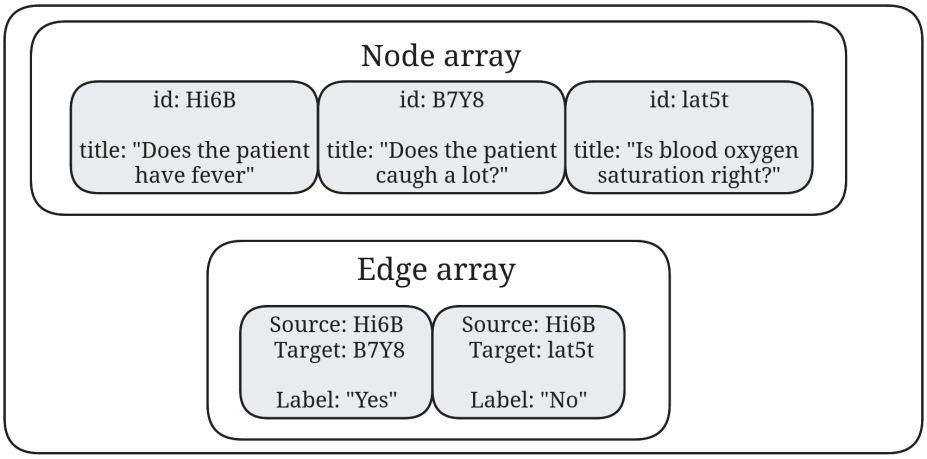

Figure 5.4: Flowchart internal representation

State management solutions

Apart from React Flow, another library which has been very helpful in the development of the editor has been Zustand [10]. Zustand is a state management library for React. The reason why this kind of library was used is the need of updating the flowchart editor state or reacting to a change of it from different parts in the component tree which contain the components which comprise the protocol editor.

React has its own solutions for state management [11], but had I used them, the code would have been very hard to understand and thus maintain.

The core of the problem is that different parts of the state belong to different components. Zustand solves this problem by introducing a global store in which the protocol editor state lives. This store can be accessed from every component. However, a component does not access all the state but only those parts which it needs. Otherwise, every component which made use of this store would be re-rendered every time a portion of the state changes and, as a consequence, performance would be negatively affected.

Zustand was chosen over other third-party state management libraries such Redux [12] because of its simplicity. Other libraries of these type are known for its complexity and opinionated way of doing things. Moreover, React Flow uses it internally and it is the one which is recommended in the docs. The main downside is that, being a relatively new library, there

[9] `https://www.npmjs.com/package/nanoid`
[10] `https://zustand-demo.pmnd.rs/`
[11] `https://react.dev/learn/managing-state`
[12] `https://redux.js.org/`

were not that many resources compared to, for example, Redux. However, my use case was simple enough that the official documentation and some articles on the topic allowed me to successfully integrate Zustand into the project.

Zustand has only been used in this page. The other pages employ the state management solutions provided by React.

Rules for a valid protocol

In order for the protocol to look right when displayed by the XR application running on the Meta Quest 3, I require the protocol to be in a valid state. Specifically, these are the rules that every protocol must satisfy so that it can be saved in the database:

1. There must exist an initial node.
2. The name of a node must be non-empty.
3. If a node has more than one outgoing edge, then the label of each of those edges must be non-empty.
4. Two edges which share the same source node cannot have the same label.

The editor checks these rules every time it is modified and when it detects that at least one them is violated, it informs the user by changing the UI accordingly. Specifically, when rules 3. and 4. are not met, the border of the `div` which contains the label turns red and when rule 2. is broken, the text *Undefined Node* appears in a red background. Rule 1. cannot be broken because the user cannot delete the initial node by pressing backspace. This state can be seen in figure 5.5.

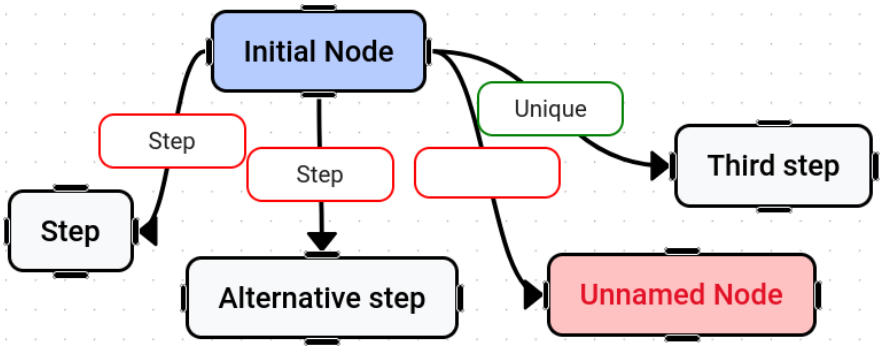

Figure 5.5: Example of an invalid protocol

Saving local state

One of the hardest parts of this project has been syncing the local protocol state with the one which is stored on the backend. The main difficulty comes from the fact that I require the protocol to be in a valid state, but when the user modifies it, it has to necessarily go through a state which I consider invalid.

There exist many solutions to save the protocol, the simplest of which consists in creating a button which the user presses when they want to save it. In case the protocol is in a invalid state, that operation is simply not performed. The main drawback is that the user may forget to click on the save button and may lose all the steps and options added since the last time they saved it. To avoid that situation, another solution is to save the state every time it changes. However, that is happening continuously if the user is modifying the protocol, specially if they are changing the position of a node by dragging it. As a result, a lot of requests would have to be sent to the server. Since that aggressive autosaving did not wholly appeal to me but I did not want the user to lose its progress, I decided to implement an autosaving functionality which saves the state after a set of changes have been detected for a period of time.

Specifically, my approach is the following. Every time a node or an edge has one or more of its properties modified, it is recorded. If more than two seconds have passed since the last time a change was recorded, then it is saved and, after that, those nodes and edges are discarded. Only the nodes or edges which have changed and thus have been recorded are sent to the backend to be saved. In other words, not all the local state is sent to the backend so that it replaces the old one stored in the database by the received new one but only those nodes or edges whose properties have been modified. For example, suppose I modify the state of the protocol by moving a node. This results in the node being recorded. After six seconds of not detecting any change, the whole node is saved, even though its position was only changed. A higher level of granularity would have made the code on the frontend and the backend more complex and the benefits were not clear to me given that I store no more than six properties for each node or edge. A high level overview of this algorithm is shown in figure 5.6.

In addition to that, if more than ten seconds have passed since the first node or edge was recorded and the state has not been saved yet, then it is saved. This is done to avoid long periods of times in which changes have been made but the user continues modifying the protocol.

When a node or an edge is deleted, that change is immediately saved, and does not affect whatsoever the process defined before. This was done mainly because it was simpler to implement.

There is one problem with this approach and it is that it is likely that, when it is time to save the protocol, it may be in a invalid state which the backend should not accept. The easiest solution would have been to abort the saving operation but that would have conflicted with the objective of the autosave functionality of not having long periods of time in which changes have been made but are not committed. The solution that I came up with consisted in saving those parts of the state which have changed but did not make the state which is already in the backend invalid. For example, consider the protocol in figure 5.5. The node without

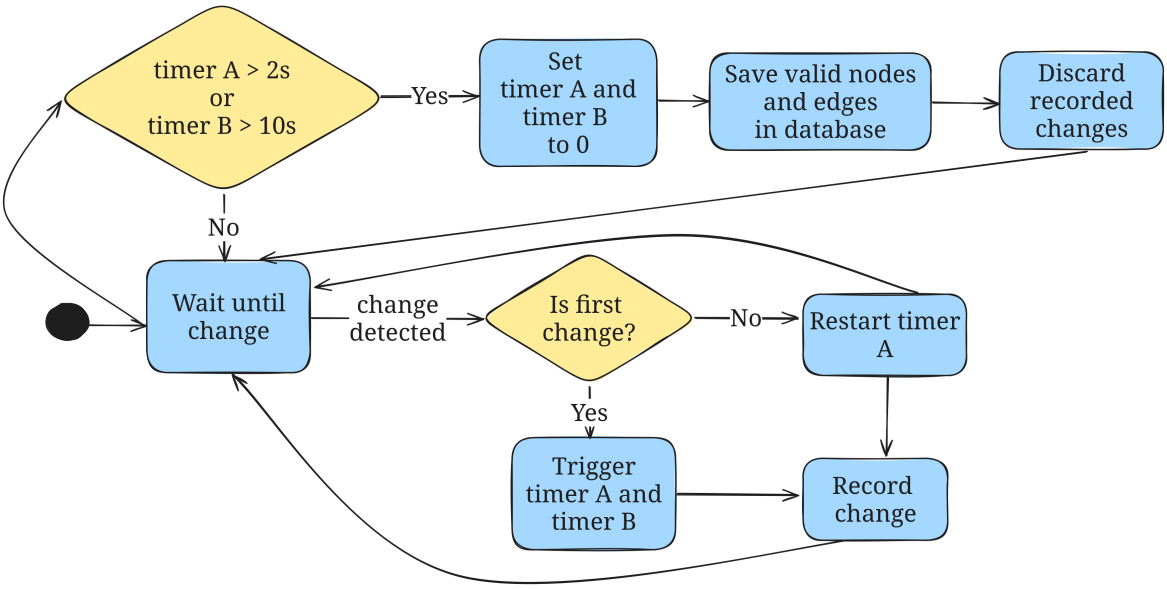

Figure 5.6: Flowchart representing the algorithm which determines when to save the protocol state

a name will not be saved and, as a consequence, neither the edge which joins it the Initial Node. The edges whose labels contain the text *Step* will not be stored on the database. If the edge had not had another saved label, then the whole edge is not saved. As a consequence, a protocol can be saved or partially saved and this situation is shown in a message located in lower right corner of the editor. This message changes to inform the user when an update to the protocol local state has happened and it is shown until the valid changes have been committed.

Other technical considerations

React Flow is DOM-based[13], which basically means that it uses the DOM to draw the diagrams. The advantage is that you do not have to deal with low level graphics primitives to show the UI elements on screen but it is not as performant as the canvas API to create graphics. Unless you are rendering a lot of content in the diagram, this is seldom a problem. However, it is important to keep this fact in mind because performance could be affected if you are displaying heavy DOM nodes. I had had performance problems because of the TextField MUI component. On previous versions of the project, it was shown on every edge and it contained the edge's label. However, when five or six of those elements had to be rendered, the editor started to become laggy. I solved this problem by displaying the label inside a `div` and only showing the TextField when the user double clicks on the edge since that represents their intention of modifying the label text. When the user clicks away, the TextField disappears and the `div` is shown again. As a consequence, there is at most one TextField rendered given at a given time.

[13] `https://github.com/xyflow/xyflow/issues/3044#issuecomment-1541584718`

5.2.2 LLM page

The objective of this page is interacting with the self-hosted LLM. Although the UI may induce the user to think that they could chat with it, this is actually not possible. This is because previously sent messages are not saved and, as a consequence, the model cannot be asked about other parts of the conversation. Even though implementing that feature would no have been extremely hard if the messages are kept client-side and they are sent as context with the new prompt to the model, it would have deviated the focus from the actual purpose of the LLM in this application, which is answering questions given only the knowledge gained during training and fine-tuning or the one contained in the documents uploaded by the user for a protocol.

There exist two variations of this page. In one variation, the answer to the user's question does not employ the knowledge extracted from any document but only the one which is encoded in the model's weights. It is accessed by clicking on the *LLM* button located on the side bar. The other variation makes use of the knowledge contained in the documents. Moreover, that view allows the user to choose the generation mode (see section 5.4.2) for synthesizing the answer. This view is accessed by clicking on the option *Chat* which appears in the menu which is displayed when you click on the button located in the upper right corner of a protocol card, in the protocol list page. The documents consulted during the generation of the answer are the ones associated to that protocol.

Receiving LLM responses

One of the most challenging parts that I have faced when coding this page has been showing the LLM response in a streaming fashion. That basically means showing the different chunks an answer is comprised of as soon as they are received by the browser so that it is incrementally displayed to the user. I could have opted for buffering the answer and just display it when the LLM had finished generating it but that would have affected User Experience (UX) because the user would have had to wait a considerable amount of time until the full response had been fully synthesized. Moreover, after the release of LLM powered chatbots, such as ChatGPT or Gemini, people expect to see the model response gradually displayed when interacting with these kinds of models.

In order to support this streaming functionality, I considered using Server Side Events (SSE)[14]. This feature allows clients to receive real time updates from a server without having to request them previously. This is achieved by first establishing a connection with the server and keeping it alive. This option seemed attractive because JavaScript offers an API to work with these kinds of events , which allows you to attach a callback every time an event of some kind is received. Regarding my problem, an event would have been the generation of

[14] `https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events`

a part of the model's answer (i.e., the generation of a token).

However, there were two details which deterred me from carrying out the implementation of this approach. Firstly, as it has been said, this method requires a persistent connection. Therefore, when a prompt had to be submitted, a request had to be made to the server so the LLM starts generating a response. But the different chunks in which the LLM response is divided would not come from the connection created for that request but from the persistent one opened to support SSE. This makes the code on both the frontend and the backend more complex and difficult to reason about.

The solution which I finally implemented makes use of the Stream API. This is a JavaScript API which allows the developer to start processing chunks of media such as video, image or text before the content has been fully downloaded [15]. Additionally, this API supports transforming one type of stream into another one. This is needed in my case because the response is being received as a stream of bytes which represent text. Specifically, the text is UTF-8 encoded so the first step is to convert the stream of bytes into a stream of text. This is done by using an implementation of a `TransformStream`. JavaScript already provides this implementation, called `TextDecoderStream`. Since the format of the responses is in nd-json (see section 5.4) the text stream has to be split on the newline character (`\n`). That results on pieces of text which are valid JSON which are then deserialized into a custom object called `LLMResponse`. `LLMResponse` has a single string property called `text` which contains a chunk of the answer. The `TransformStreams` which are in charge of that transformation were inspired by a snippet of code hosted in a Github gist[16]. A graphical explanation of this process can be seen in figure 5.7.

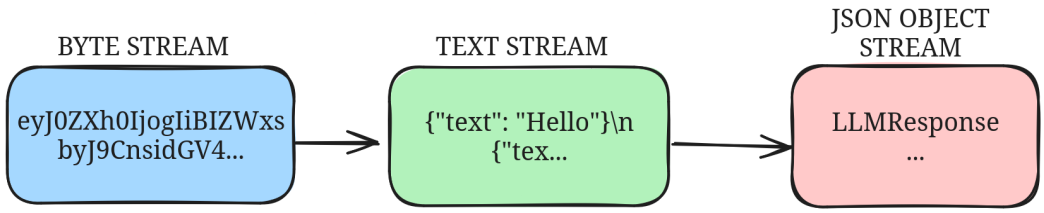

Figure 5.7: Byte stream to object stream representation

Therefore, instead of reading from the byte stream, I read from the `LLMResponse` stream, whose content is then concatenated with the one which was previously read from it.

5.3 The protocol service

The aim of this service is to persistently store the data which comprise a protocol, the images attached to a protocol step and the documents associated to a protocol.

[15] `https://web.dev/articles/streams`
[16] `https://gist.github.com/nestarz/1fa7ae93fb83f1eafb1b88c3a84f2e02`

Since all of those elements need to be retrieved by two devices so distinct such as a web browser and the Meta Quest 3, a JSON API was created to interface with the storage software. In this way, the functionality offered by this service can be consumed by both of them by performing HTTP requests. The frameworks which have been used to develop the application for each device contain an HTTP client capable of performing those operations as well as mechanism for serializing or deserializing the content of the messages which are sent or received.

The endpoints this service exposes are mostly CRUD, because they create, read, update or delete one or many of the aforementioned elements.

The framework which has been used for the creation of the JSON API is FastAPI[17]. It is written in Python and integrates tools which make the incorporation of recurring elements in a JSON API very easy, such as data validation. Moreover, it automatically supports the generation of documentation in a standardized format known as OpenAPI[18]. FastAPI is built on top of Starlette, an HTTP server implementing the Asynchronous Server Gateway Interface (ASGI) [19]. As a consequence, the handlers for an HTTP request can be co-routines so the whole server can run on a single thread and still concurrently serve multiple requests. The main drawback of writing asynchronous code is that the used libraries must have support for it. Otherwise, the event loop can get blocked and performance is affected. Although there exist versions or alternatives to the libraries I have used which are async-aware, for some of them there were not as much documentation compared to the version or alternative employing blocking calls. As a consequence, I opted for writing the handlers as normal functions. In that case, FastAPI runs them in a thread pool, so blocking calls are not a concern.

Other frameworks were considered but I finally opted FastAPI because I had used in the past and I have experience programming in Python.

5.3.1 Saving a protocol

The data which is necessary to show a protocol in the flowchart editor or in the visualizer developed for the XR application is stored in a relational database called MariaDB. MariaDB is a fork of MySQL. Initially, MySQL was chosen because I had worked with it before and, therefore, was already familiarized with it. However, it lacks an SQL extension which I needed for some queries. Specifically, I was interested in the RETURNING option in a INSERT statement [20], which returns the records which have been added. This is very useful if the table contains an auto-increment column because you are able to get the value assigned to them after they are inserted. As a consequence, I opted for the most similar relational database which included that extension, which was MariaDB.

[17] `https://fastapi.tiangolo.com/`
[18] `https://swagger.io/specification/`
[19] `https://asgi.readthedocs.io/en/latest/`
[20] `https://mariadb.com/kb/en/insertreturning/`

I also considered PostgreSQL but, since I was going to use an ORM, I would not have used its distinguishing features because part of the objective of an ORM is to provide an uniform interface over all relational databases so that you can easily retrieve objects from it. I would have needed to step outside of the functionality offered by the ORM. However, none of those features were of interest for me for this project. Therefore, I decided to stick to a Relational Database Management System (RDBMS) I had used in the past.

The Entity–Relationship (ER) diagram is shown in figure 5.8.

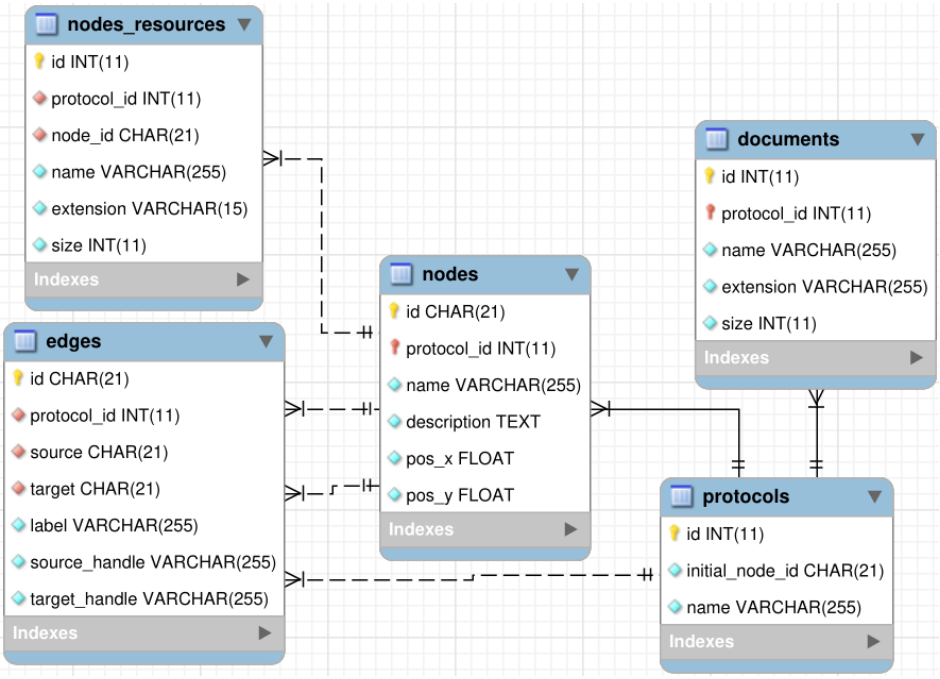

Figure 5.8: ER diagram
ER diagram of the entities stored in the relational database.

Considering a document oriented database

Before choosing a relational database, I analyzed whether a NoSQL database would be a good fit for my use case. Specifically, I focused on document oriented databases, such as MongoDB or CouchDB. Documented oriented databases are a good option if the relationship between the entities which form part of the domain follow a tree like structure (tree of one-to-many relationships) and there do not exist many many-to-many relationship . As a consequence, related data is kept close to each other and not spread across different tables, as it occurs in the relational model [Kle17]. Moreover, these kind of databases are more flexible in the sense that you do not need to define a schema which determine the structure of the data. Therefore, a change of the schema is easily manageable, at least by the database. This is because inserting data is reduced to just dumping a JSON string.

Even though there do not exist many-to-many relationships in the ER diagram shown in 5.8, I did not know that at the beginning of the project. Moreover, I also had to take into

account the time spent learning a new piece of technology with which I had no experience such as a document oriented database. Regarding schema flexibility, MariaDB supports saving JSON which can even be queried [21].

The ORM

The ORM I have used is SQLAlchemy because it is the most popular for Python and, therefore, there exist a lot of information about it on the Internet. An ORM tries to reduce the *impedance mismatch* which exist between the relational model and the object oriented paradigm. However, an ORM is leaky abstractions because you need to know how the relational model and SQL works to effectively use it. In fact, given the low complexity of the queries which had to be performed, they could have been written in raw SQL. However, since I opted for the ORM, time had to be spent translating the raw SQL queries into the format which accepts SQLAlchemy.

5.3.2 Storing documents and protocol step resources

The document associated to a protocol and the images attached to a protocol state are stored on the VPS file system. I initially thought on saving them in the relational database but after some research, I found out that saving a lot of them could affect performance. Moreover, I wanted those static resources to be served by web server specialized in that task, such as NGINX. For that, the best option was to save them in the file system.

Before storing a file to the file system, it will check that is of the correct type. Specifically, only png and jpeg images are allowed because those are the formats which Unity supports by default. For documents, pdf and UTF-8 encoded txt files are supported because those were the most straightforward formats to process. The modules in the global architecture diagram in figure 5.1 in charge of that task are *Image file validator* and *Document file validator*.

When an image is uploaded to a protocol step, its name is substituted by a unique identifier and stored in one directory in the file system. The file extension must be kept so that the web server associates a correct Multipurpose Internet Mail Extensions (MIME) type to the content which is serving. All images are stored in the same directory, even though they may belong to different protocol steps or even to different protocols. The data which links back an image to a protocol step is stored in the relational database. Additionally, its name, size and extension is saved along it. That is the purpose of the tables `nodes_resources` and `documents` in the ER diagram of figure 5.8.

The same strategy is followed for a document. The only difference is that the LLM service is contacted so that that it is processed and stored in a format which is quickly retrievable for the generation of an answer. This can be seen in the diagram shown in figure 5.9.

[21] `https://mariadb.com/es/resources/blog/using-json-in-mariadb/`

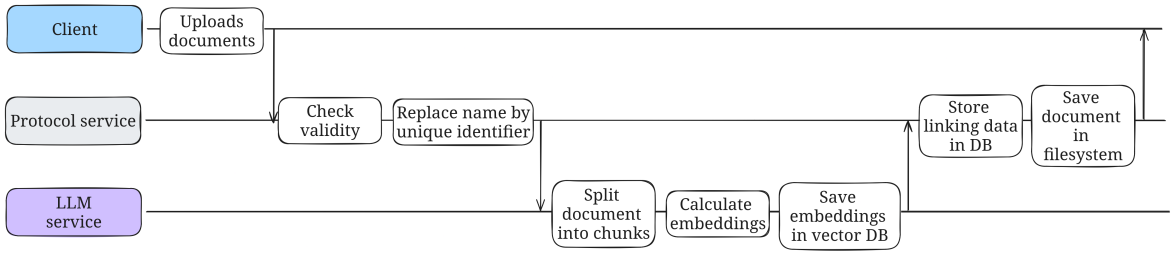

Figure 5.9: Sequence of steps performed when a document is uploaded

5.4 LLM service

This service is in charge of querying the self-hosted LLM for an answer to a specific question as well as processing the user uploaded documents in such a way that they can be stored in a manner such as their content can be quickly retrieved for the generation of the LLM response. Since the functionality of this service is meant to be consumed by processes running on another machine, a JSON API has been created. FastAPI was chosen for that task for the same reasons as it was selected for the protocol service (see section 5.3)

This JSON API exposes four endpoints but three of them are designed to be called only by the service running on the VPS. Those three are the ones in charge of creating or deleting a document or all the ones associated to a protocol. This is because the responsibility of processing a document is divided between the LLM service and the one running on the VPS. While the former stores some metadata of it in the relational database and saves the actual file in the VPS file system, the latter splits it into chunks, calculates for each of them an embedding vector and store each embedding-chunk pair on a vector database

The other one, which is `/llm/generate`, is meant to be contacted directly and it is in charge of generating a response given a prompt. It supports synthetizing an answer from the *knowledge* encoded in the weights as well as using the information contained in the chunks of texts stored in the vector database to provide more accurate responses. This is controlled with two query parameters. Specifically, the query parameter `protocol` specifies the id of the protocol whose documents are going to be used and `mode` establishes the generation which is going to be employed. The response to a request to that endpoint is in a format called nd-json. nd-json consist of JSON objects separated by line breaks (`\n`). It is thought for streaming JSON objects over a TCP connection [22]. This format was chosen because the partial responses (i.e., the tokens) generated by the model are serialized into JSON and then streamed to the client using the connection which was opened when the HTTP request was made. Other formats could have been used but this was the simplest one.

The reason why the vector database is not running on the VPS is that the model which generates the embeddings needs to be executed on a system with a decent GPU in order to obtain the results in a reasonable amount of time, which is something the VPS lacks. I could

[22] `https://github.com/ndjson/ndjson-spec`

have offloaded the computation of the embeddings to a more powerful machine, such as the AIR research group server, but this solution would have been non-trivial because that would have lead to the implementation of the RAG process from scratch, apart from implementing that offloading functionality. This is because I could not find a way of correctly fitting the library which facilitates incorporating knowledge into the answers into that setup. Either way, hosting the vector database on the VPS would remove the inconsistencies that may appear for having to keep different kinds of information for a document synced between two machines whose communication can fail at any moment.

5.4.1 Running the self-hosted LLM

As its name implies, an LLM is a large model which requires demanding hardware resources to run, specially an important amount RAM and VRAM. There exists a technique which has helped enormously in the objective of executing a LLM on consumer hardware, known as quantization [FAHA22].

In the context of LLMs, quantization refers to the process of reducing the precision of the weights so that the memory footprint during inference is not that big. As a result, the weights are not stored using 32 bit floating point numbers, which is the data type normally used for training, but other lower precision types such as 16 bit floating point or even 8 bit integers. The quantization process also speeds up inference since the time required by the GPU to perform the operations on lower precision types is smaller than that required for the ones employing more bits. The challenge of quantizing a model lies on reducing the weights precision without sacrificing too much the LLM text generation quality.

There exist many quantization algorithms but each one falls under one of two categories [FAHA22]. The first one refers the quantization techniques which are applied after the model has been trained with full precision. These are the easiest to apply but model performance may be affected because the posterior reduction in precision is not taken into account during training. This contrasts with methods which consider the quantization process since model training. As a consequence, they usually perform better but training them becomes more difficult. In my case case, I have chosen a model which has been quantized using a technique which belongs to the former category because it is more convenient to apply.

In order to run a quantized model, the implementation has to be aware of the quantization techniques employed. A popular software for running quantized models is llama.cpp[23]. Its main goal is to execute a model on CPU but it supports offloading part of the layers to the GPU. It is written C/C++ and it uses a tensor library called ggml developed by the same author. This is where the actual support for quantization is implemented. Although it is called llama.cpp, it supports other LLMs outside of the LLaMA family of models and the list available models is continuously growing. llama.cpp also includes a C API so that it can

[23] `https://github.com/ggerganov/llama.cpp`

be included in other software or bindings for other programming languages can be created. Since the model needs to be called from Python, the Python bindings were needed. Those which have been used are the ones provided by the Github user abetlen [24].

Since *airproy* has more than enough VRAM to completely hold the model, all the layers have been offloaded to the GPU. The model is loaded once and it is kept in VRAM as long as the service is running.

The Python llama.cpp bindings can only generate one response at a time. For that reason, if the Python method from the bindings which triggers the generation is called from another thread when llama.cpp is in the middle of the process of answering a query, the program will crash. Although there exist techniques which support the concurrent generation of responses, this is done by reducing the context window size for each one and I was not interested in that. This is because that would lead to a decrease in the amount of information the LLM considers to generate the answer. Since the system was designed to be used by a single person at a given time, I opted for protecting the call to the generating method with a lock. As a consequence, every time the endpoint `/llm/generate` is contacted, the lock is acquired. If it is already acquired, a Service Unavailable error is returned to the client. The lock is only released when the generation finishes or an error occurs during that process.

Choosing the model

The chosen LLM has been Mistral 7B-v0.2 with the llama.cpp quantization method Q5_-K_S. The downloaded weights corresponds to the fine tuned version and were obtained from a HuggingFace repository, whose owner is known by the name *TheBloke*, which specializes on quantizing the weights of different models using different methods and formats. The quantized weights have a size of 5.00 GB and the maximum RAM the quantized model could use is 7.50 GB. This contrasts with the 16 GB of RAM needed to run the full precision model [25].

These weights were initially chosen because enough RAM was available in my computer for that model to fit, so all development could be made locally. I thought on running a larger model like Mixtral 8x22 on *airproy* because larger models tend to perform better than smaller ones but, even though it can run the quantized method, it was not as fast the first option. Since I obtained decent results with Mistral 7B, that was the one which was finally deployed.

Even though llama.cpp contains scripts which allow you to quantize the full-precision weights, I found more convenient to download the weights already prepared for the inference.

Mistral 7B was chosen over models of comparable size such as LLaMA 2 7B or LLaMA

[24] `https://github.com/abetlen/llama-cpp-python`
[25] `https://docs.mistral.ai/getting-started/open_weight_models/`

2 13B because the one developed by Mistral is truly open source since its use is governed by the Apache 2 license. Other open-weight models put restrictions in the way you can use the generated text. Moreover, Mistral 7B has been trained on content under the public domain or with licenses which allow that type of use [26]. The main drawback of this election is that Mistral 7B does perform poorly when it is asked in a language other than English.

5.4.2 Retrival Augmented Generation

Retrival Augmented Generation (RAG) is a methodology used for incorporating information from an external knowledge source into the responses of a LLM. RAG defines three steps [GXG+23] to achieve that:

1. **Indexing**. The first step consists in extracting the text from the different documents and possibly clean it. The text is then divided into chunks because the LLM context window limits the size of the prompt so passing all the documents would not work. These chunks are normally overlapping. Finally, an embedding vector (see section 5.4.3) is calculated for each chunk and the pair embedding-chunk is stored in a vector database. Embeddings are used because they allow the retrieval of text chunks based on their semantics.
2. **Retrieval**. The query is transformed into an embedding vector using the same procedure used to obtain the embeddings for each chunk. Then, a specified number of chunks whose embeddings vectors are similar to the ones obtained from the query according to a similarity metric (normally the cosine similarity) are retrieved. These chunks will be used as expanded context to the prompt.
3. **Generation**. The final step consists in creating a prompt using the query and the retrieved chunks which is passed to the model for the generation of the responses. For the process of synthetizing a prompt, a template is normally used.

A diagram of that procedure can be seen in figure 5.10.

RAG is not the ultimate solution for incorporating knowledge into an LLM. There exist problems which may affect the output quality of the model. One challenge is the difficulty of selecting the relevant chunks for some kinds of queries. A careless implementation of this phase may lead to selecting irrelevant chunks. Additionally, it is not guaranteed that the model will not make up some facts, a phenomenon known as hallucination. However, the probability of hallucination is lower compared to to not using the RAG methodology.

To avoid implementing this methodology completely from scratch, I decided to use a library which already provides building blocks to create RAG based LLM agents called LlamaIndex[27]. I was specially interested for its tools for transforming text into chunks which

[26] `https://mistral.ai/news/announcing-mistral-7b/`
[27] `https://www.llamaindex.ai/`

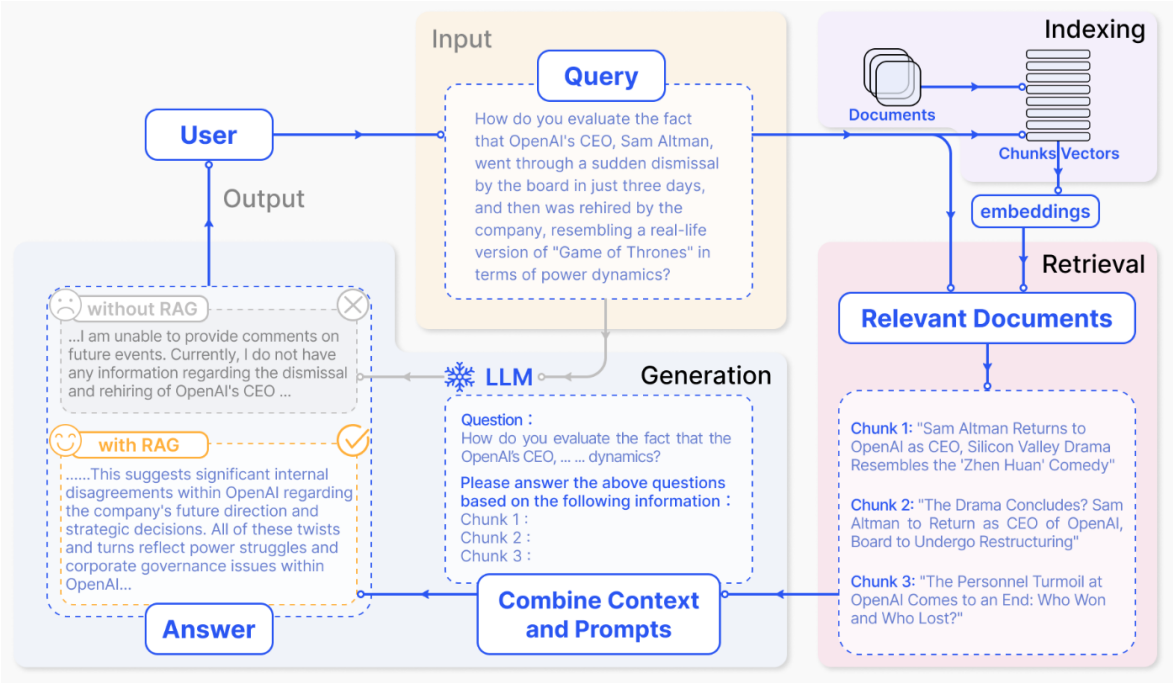

Figure 5.10: Depiction of the RAG process
Diagram obtained from [GXG+23]

can be stored in the vector database. However, it was also used for the retrieval and generation phases since it is very difficult to only use the tools for one phase without employing the ones for the others because of the dependencies LlamaIndex creates between the three of them.

Generation modes

LlamaIndex supports several generation modes, but two have been used. One concatenates all the chunks and passes them to the model. For that reason, it is known as concatenate mode. The other is thought for questions which require multiple steps to be answered. For example, the question *Who is the president of the country where the 2024 Olympics are held*? requires to first now the country to be able give the president's name. In this generation mode, LlamaIndex retrieves the relevant chunks and pass each one individually to the model, with the information which has been obtained from the model in a previous response also included in the prompt. As a consequence, this multi-step mode requires more calls to the LLM but the answers may be more accurate.

The templates used by LlamaIndex to generate the prompts from the chunks of text extracted from the vector database are shown below.

The original query is as follows:
QUERY
We have provided an existing answer:
EXISTING_ANSWER
We have the opportunity to refine the existing answer (only if needed) with some more context below.
CONTEXT_MSG
Given the new context, refine the original answer to better answer the query. If the context isn't useful, return the original answer
Refined Answer:

Multistep mode prompt template

Context information is below.
CONTEXT_MSG
Given the context information and not prior knowledge, answer the query.
Query: **QUERY**
Answer:

Concatenate mode prompt template

5.4.3 The sentence embedding model

Vector embeddings are not only obtained from words but they can also be generated from sentences. In order to obtain sentence embeddings, models designed for that task are used. These embedding vectors can be used in downstream tasks but they can also be compared to obtain a similarity score. One metric which is very popular is cosine similarity [SEK24]. That score reflects the semantic similarity that exist between the two compared sentences. Since calculating the similarity score between two vectors is a fast operation, embeddings are a good option to quickly retrieve chunks of information related to a question since one expects a question is somehow related to its answer and that fact should be reflected in the similarity score.

The chosen embedding model is bge-large-en-v1.5 [28] and it is also self-hosted. The main reason why I chose it over other options is because it is one of the best performing ones in retrieval tasks compared to other of similar size[29]. The embedding dimension is 1024 and the maximum sequence length is 512. That last number determines the maximum size of the sentence which can be given to the model and, as a consequence, determines the size of the chunks in which documents are splitted. The model weights 1.4 GB.

5.4.4 The vector database

In order to efficiently retrieve the chunks whose embedding vectors are similar to another one according to a similarity metric, vector databases are used. Vector databases are a not a new piece of technology but have recently become more popular as a consequence of

[28] `https://huggingface.co/BAAI/bge-large-en`
[29] `https://huggingface.co/spaces/mteb/leaderboard`

RAG.

There exist many options but the one I decided to use was Qdrant[30]. This was because Qdrant is supported by LlamaIndex and provides a Docker image which makes it easy to self-host it.

Qdrant has been configured to efficiently process the types of queries it receives. All of them ask for the top K most relevant chunks from a specific protocol. The configuration allows the database to be faster at carrying out the filtering of the chunks by the protocol. This has been possible by enabling a feature called multi-tenancy. [31]

5.5 XR application

This is the application which is running on the Meta Quest 3. It is in charge of showing the different steps a protocol is comprised of as well as displaying helpful images associated to each of them. All of that information has had to be previously introduced by the user using the web application. Moreover, it makes use of a speech-to-text model which transcribes an user's question into a piece of text which can be sent to the LLM service. In order to improve the UX, the generated answer is sent to a text-to-speech model to be read out loud.

The application supports medical staff by telling them the steps they should follow to perform a correct diagnosis and recommends an appropriate treatment. That extra help will be given to the professional without limiting the range of movement they can do with their hands because the only device needed is the Meta Quest 3. The Quest 3 controllers are not needed for interacting with the application because the headset performs hand tracking. As a consequence, other type of medical equipment, such as sphygmomanometers or oximeters, can be handled without leaving the environment in which the user are receiving the information.

5.5.1 Developing for the Meta Quest 3

For the development of the application, the 3D real time engine Unity [32] was used. Even though other engines like Unreal Engine support developing XR applications, it has a steeper learning curve than Unity. Additionally, Unity has a big community and, as a consequence, there exist a lot of information on the Internet about it. Finally, I already knew the basics of working with it, so I could start creating working results faster than with another alternative.

Unity is a multi-platform program because it can run on Windows, GNU/Linux and Mac OS. In spite of being my favorite Operating System (OS) for developing software because of the large amount of available tooling to customize you workflow, GNU/Linux was discarded

[30] https://qdrant.tech/
[31] https://qdrant.tech/documentation/guides/multiple-partitions
[32] https://unity.com/es

and instead Windows was chosen. The rationale behind this decision is that the GNU/Linux Unity version is not as polished as the Windows one. Additionally, the software which allows you to connect the headset to a computer without any problems only has a Windows version.

Specifically, that software is called Meta Quest Link and it is a requirement [33] if changes to the Unity application want to quickly be tested on the headset without having to wait until it is recompiled and uploaded to the device. Instead, with Quest Link, feedback is received in a few seconds after the *Play* button is pressed in the Unity editor. The main drawback of this way of testing the application is that the headset pass-through functionality, which allows the user to see the images captured by the Meta Quest 3 cameras as their environment, does not work. The only solution is to build the application and run it on the headset. However, since the cameras were not used for any purpose other than adding pass-through to the application, that limitation was only a minor inconvenience. Another consideration the person developing for the Meta Quest 3 should keep in mind is that the headset should be connected to the computer using an USB 3.0 type C. Using an USB 2.0 may lead to unexpected behaviours.

Choosing an AR library for Unity

There exist libraries for Unity which provide building blocks to create MR applications in Unity. They save the developer the time they would have spent creating different UI elements using the primitives Unity makes available by already implementing them. They also support attaching an action when specific gestures are detected. Some libraries are only targeted to a specific headset like the Meta XR Core SDK, which is focused on Meta Quest devices. Others, like Mixed Reality Tool Kit (MRTK), can be used with a wide variety of headsets.

I was recommended to use MRTK 3 [34] because, given that the library supports several headsets, the code could be ported to other devices with possibly minor changes. However, although I managed to setup a MRTK 3 project which could be loaded to the Meta Quest 3, the pass-through could not be enabled. The reason why I think it is the case is because MRTK 3 is able to target different devices because they implement the OpenXR standard, which is the case for the Meta Quest 3. However, the standard does not include pass-through as part of its supported features. As a consequence, there does not exist a defined way of accessing that functionality in OpenXR compliant devices. There exist, nevertheless, a vendor extension to the standard proposed by Meta to allow that. However, at the time of writing this document, it is only implemented by Quest and PICO headsets [35]. I suspect that MRTK 3 is ignoring that extension.

[33] https://www.meta.com/es-es/help/quest/articles/headsets-and-accessories/oculus-link/set-up-link/

[34] https://github.com/MixedRealityToolkit/MixedRealityToolkit-Unity

[35] https://docs.godotengine.org/en/stable/tutorials/xr/openxr_passthrough.html

The possibility of using MRTK 2 was analyzed. MRTK 2 supports Quest devices by means of the the Oculus integration SDK, which was developed specifically for Quest devices. Since MRTK 2 is interacting with the Quest device using building blocks specifically developed for it, adding pass-through was as easier as adding the *pass-through building block* to the application. However, the Oculus integration SDK was deprecated in favour of Meta XR Core SDK. Since starting a new project with deprecated libraries was not appealing to me, I discarded it. I also considered using the building blocks provided by the Meta XR Core SDK. However, I also rejected this option because code would not be portable. Moreover, there were not as much documentation compared to the one which exist for library I finally decided to use.

In the end, I settled for the XR Interaction Toolkit, developed by Unity. As it occurs with MRTK 3, XR Interaction Toolkit targets OpenXR compliant devices. XR Interaction Toolkit works with the plugin management system Unity has (see section 3.3.2). I decided to use OpenXR plugin, thinking that, in that way, the application could run on another headset. However, since it was necessary to enable OpenXR vendor specific features, such as the one which makes pass-through available, the code will not work on another device unless it implements those extension features. Although it does not include as many features for developing AR applications like MRTK, I considered it to be enough for my use case. Moreover, it integrates very well with the tools Unity has for developing UIs. As a consequence, all the documentation there exist for those tools can be useful for the development of the application.

Instead of creating a new project from scratch and then adding the XR Interaction Toolkit, I opted for starting off from an official Unity template project which is already configured to start developing XR application with that library.

5.5.2 Development

The information is displayed in floating canvases (see figure 6.6). The functionality of each canvas is contained in a Unity script, which are all attached to an empty game object located at the root. A script references other scripts when they contain functionality which may result in the appearance of another canvas. When a canvas appears, a method defined on the script in charge of its functionality is called. This is the way in which information on one canvas is shared with another. This tree of script references is shown in figure 5.11.

The `UIManager` script has been used to place the functionality that did not exactly belong to `ProtocolManager` but neither to `ProtocolListManager`. `NodeResourcesManager` is the script which manages the canvas which shows the images associated to a step. It was named in that way because I thought other types resources were going to be attached.

For example, in figure 5.11, the code which handles the event of pressing the button which shows the images associated to a step is found in `ProtocolManager`. That code calls the

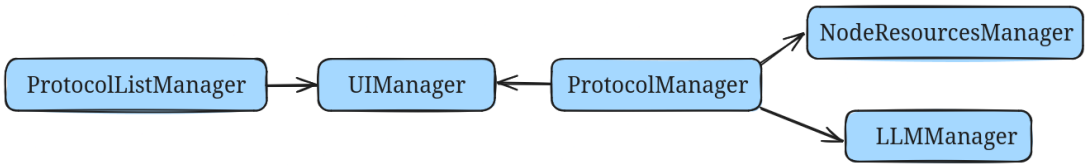

Figure 5.11: Tree of Unity script references

method `OnVisualizerShown()` defined in `NodeResourcesManger` which receives a collection of the URLs which point to the images.

When a protocol is launched, the graph which represents the flowchart is received as a response to a JSON API call in the format outlined in figure 5.4 However, working with that representation in this application was very inconvenient because, for every node, I want to quickly access its adjacent edges. For that reason, the class `ProtocolFlow` was created. This class is initialized with the inconvenient representation and defines methods to make it easier to handle them for this use case. It is also in charge of keeping the state of the protocol execution (i.e., the current step and the previous ones) although it would have been probably better to separate those two functionalities.

The user is able to *speak* to the model and the model *speaks* back to them thanks to the NLP models provided by wit.ai [36], which is a company owned by Meta. It is specialized in creating the building blocks NLP powered applications. The Voice SDK Unity package allows you to easily integrate the services provided by the company into an Unity application and it is the one which has been used. Specifically, the company's speech-to-text and text-to-speech models have been employed. A problem with the latter one is that you cannot send a piece of text with more than 140 characters all at once. For that reason, the LLM response had to be splitted on pieces with at most that number, with the extra difficulty of having to perform that division at word boundaries.

Furthermore, the application performs hand tracking so that the Quest controllers are not necessary. It is done by a module provided by XR Interaction Toolkit. Apart from enabling the user to interact with the application by using the appropriate gestures, it also allows the developer to set some elements relative the position where a hand is located. For this application, the menu showing the list of protocols appears next to the user's left hand when the palm is in front of their face. This feature appears in the global architecture diagram under the name of *Left hand tracking*.

5.6 Deployment

Some services have been deployed on a VPS rented from the company IONOS and others on the server acquired by the AIR research group for compute intensive workloads, which is referred as *airproy*. Specifically, on the former, the NextJS server, the MariaDB database

[36] `https://wit.ai/`

and protocol service are running and the latter hosts the LLM and the vector database. A layout of the system architecture is given in figure 5.12

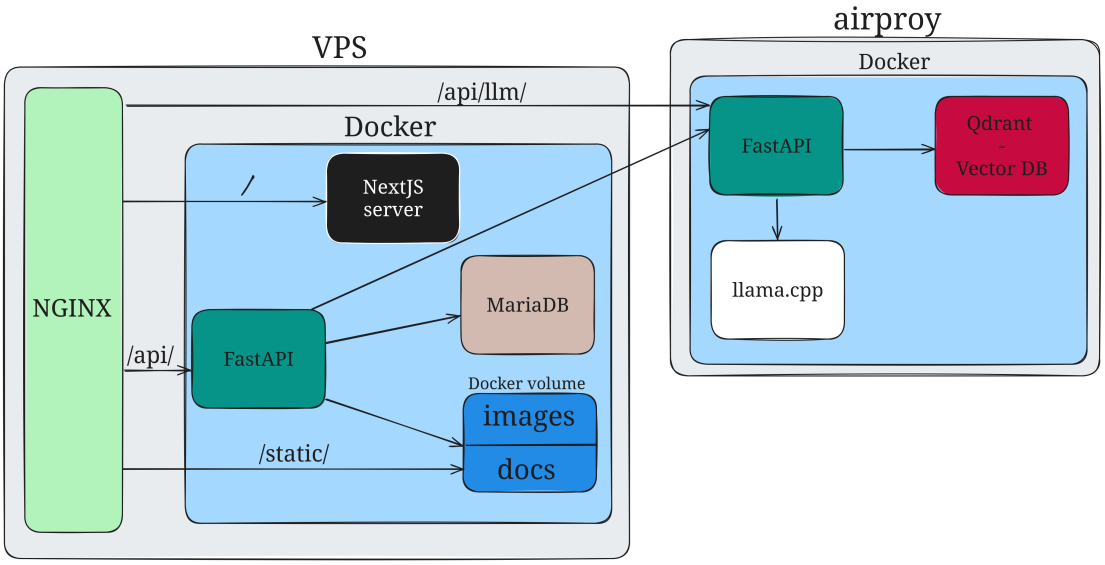

Figure 5.12: System architecture diagram

Even though the AIR research group server is completely capable of running the whole system, the application was designed to be executed on two separate machines because I doubted I was going to have at my disposal a server with the needed requirements to run the self-hosted LLM and with a public routable IP address. There were some computers which I could have used for that task, but all of them were behind NAT. Therefore, I thought on hosting the most lightweight components of the system on a VPS and forward the requests to the non-routable machine via some kind of tunnel which connects both of them. For that reason, it was very surprising to me to find out there exist plans for assigning *airproy* a public routable IP address and a domain name. However, at the time of deploying the system, the server has been assigned a public IP but traffic other than the one generated inside the university network is rejected. As a consequence of that, the system has been deployed as it was initially envisioned.

The main drawback of using a VPS or a bare metal server such as *airproy* is that I have to worry about a lot of technical details so that the deployment and posterior execution is successful. Nevertheless, there exist companies which offer platforms which abstract all of those details and make the deployment process more pleasant. They are also capable of abstracting the hardware resources, so that only the necessary ones are used to deal with the load experienced by the system at a given moment. Examples of companies which offer these kinds of platforms, known as cloud computing platforms, include Amazon (AWS), Microsoft (Azure) and Google (Google Cloud).

There are many reasons why I decided not to choose any of them. One them, which has been very important in my decision, is that it is very hard to predict its cost at the end of the

month because that mainly depends on the load their services have experienced. It is true that you can set a limit on the monthly cost or trigger some kind of alert when the expenses go beyond a certain threshold but, even with those features enabled, it is very hard to beat the VPS flat fee. Moreover, a system has to be designed from the beginning in a certain way to leverage all the services those companies provide. That way of designing software systems is knowledge which I do not possess yet, so I would not have been able to fully utilize their platform. On other hand, it is important not to forget that what has been done is closer to a research project than an enterprise-grade one so the availability and scalability these platforms provide were not something I paid to much attention to.

Apart from all of that, I wanted to have as much control as possible over the infrastructure to actually learn what those cloud-computing platform hide for convenience.

5.6.1 Getting a domain name

IONOS is not only a VPS provider but also a domain registrar. Since this was the first time I hired one of their services, I was given the option of having one domain for one euro the first year and ten euros per each extra year I wanted to have control over it. For this fee to apply, the domain had to be considered "non-premium". A chose a rather long one so that I did not fall under that category.

The web application is not hosted on the *root* domain but on a subdomain of it. In order to create it, a DNS record had to be added to the zone file. Specifically, I created a CNAME record which points to the apex domain. All of this can be easily done with the dashboard IONOS provides.

5.6.2 *Dockerizing* the services

Since *airproy* was not available during the first months, the ITSI computer was used to test different configurations for the deployment. In order to reduce number of problems which usually arise when you run software on a different environment where it was tested or developed, the different services have been containerized using Docker. Docker allows you to package a piece of software with all the dependencies it needs to run. By using Linux kernel features like namespaces and cgroups, each container is isolated from each other but they all share the same underlying kernel. As a consequence, a lot of developers usually view containers as some sort lightweight virtual machine.

Three Dockerfiles were written. Specifically, the services which needed one were the NextJS server, the protocol service and the LLM service. There already existed official images for MariaDB and Qdrant, whose environments can be modified via environment variables or by the action of another container so I considered that a Dockerfile was not needed for any them. The hardest Dockerfile to create has been the one which containerizes the LLM service. This is because it requires access to the GPU and for that, two packages have to

be installed on the system. Additionally, you have to use the Docker images provided by NVIDIA[37]. If you use another one, the application will ignore the NVIDIA GPU.

Another difficulty I have faced is synchronizing the startup of containers defined in a single compose.yml. This is necessary because there exist circumstances in which a container must be running before another starts. For example, a database must be fully initilized before it can receive requests from another container. Even though Docker Compose allows you to specify the order in which containers start, it does not know if it has been loaded completely so it is possible that a container starts before another one which it depends on has been fully initialized. The proper way to deal with this is to implement a health check mechanism which monitors the status of the application and starts the container when it detects it is ready. This can get quite complex so I opted for the most simple solution which is to wait a fixed number of seconds to leave time for the dependant systems to get ready. This approach has been implemented for the LLM service because the container which serves the JSON API to interact with the model makes a request to the vector database as soon as it starts. That is something LlamaIndex does. Unfortunately, I have not been able been able to suppress that behaviour from it.

The images are self-contained, because they include all the needed dependencies and files to run. This is also true for the LLM service, since the image already includes the LLM weights as well as the embedding model.

5.6.3 Connecting the two machines

A requirement which arises as a consequence of splitting the system between two servers is the need of both being able to communicate between themselves. The difficulty relies on the fact that the VPS is always the one who initiates the request to *airproy*. Since *airproy* is behind NAT, that is not possible by default.

Two options were considered to bypass this restriction. Firstly, Wireguard [38] was proposed. Wireguard is a Virtual Private Network (VPN) protocol which is characterized for being very easy to setup compared to other alternatives. It would have been used for the creation of a VPN tunnel between *airproy* and the VPS. The contents which traverse it are encrypted. In addition to that, a message would have to be sent at a fixed interval through it to ensure the tunnel is not dropped by the router or routers performing NAT. This keep alive mechanism is supported out of the box by Wireguard and only needs be enabled in the configuration file. However, this solution was rejected by the *airproy* sysadmin because it was deemed very "intrusive". This is because Wireguard is a L3 VPN protocol, so it requires superuser privileges to setup the tunnel since the server network configuration has to be modified and he was not willing to give to my user account more privileges. Moreover,

[37] `https://hub.docker.com/r/nvidia/cuda`
[38] `https://www.wireguard.com/`

the sysadmin would have had to add firewall rules which applied to the VPN IP associated to my VPS so that the same rules are enforced as if I were connecting to *airproy* in the normal way.

The approach which has been implemented involves opening a reverse SSH tunnel. It works by creating an SSH connection from *airproy* to the VPS and keeping it open. On the VPS, a port is bound so that the data sent to it is automatically sent through the encrypted tunnel to a port which has been specified in the command which creates the tunnel. The command which is executed on *airproy* is the following:

```
ssh -fNTq -R 8001:127.0.0.1:8000 vps
```

Therefore, when data is sent to port 8001 in the VPS, the SSH server forwards it to port 8000 on *airproy*, which is the one the LLM is listening for incoming requests. To keep the connection open, the options `-o ServerAliveCountMax=5` and `-o ServerAliveInterval=90` are used. The former establishes the maximum number of requests which has not received response before dropping the connection, which are five in my case, and the latter specifies the frequency at which a message is sent to tunnel. In my setup, that happens every ninety seconds is sent through the tunnel.

Running the aforementioned command does not require any privileges, so it has been run without the intervention of the sysadmin. The main drawback is that, in case *airproy* is restarted, I would need to type the command again. I could have created a cron job which creates the SSH reverse tunnel when *airproy* starts, but that would have required me to leave the access credentials to the VPS in it, which I did not want to.

5.6.4 Use of a reverse proxy

A reverse proxy is a server which sits in front of a web server and forwards client requests to those web servers after possibly some processing [39]. There exists many reverse proxies but I opted for NGINX because I already knew the basics of configuring it. Moreover, it is a very popular option so there exist a lot of information about different kinds of setups on the Internet.

NGINX is running directly on the VPS and not on a Docker container. This is because the VPS was meant to host unrelated applications and NGINX would act as a single entry point to all of them. Moreover, configuring HTTPS is easier if NGINX is installed directly in the system.

A reverse proxy can be used for many things, but for this project it has served three purposes. On one hand, NGINX is the component of the system which hides the implementation of the backend by providing a uniform interface to all the services. This is done by analyzing

[39] `https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/`

the request URL prefix and sending it to the appropriate service. For example, when NGINX detects the URL starts with `/api/`, it is sent to the protocol service after removing that part from it but when it intercepts one which begins with `/api/llm/`, it is forwarded to the LLM service via the SSH tunnel (see section 5.6.4). Moreover, this makes some internal endpoints on the LLM service unreachable externally.

In addition to that, the reverse proxy is the one in charge of decrypting the requests before they are forwarded to the internal services and encrypting their responses before they are sent to the client using the TLS protocol. In order to do that, the server needs to present the client a certificate signed by a Certificate Authority (CA). The CA I have chosen is Let's Encrypt because the certificate issuance process is almost automatic and it is free. By using a protocol designed by that CA called ACME, a program called `certbot` handles all the details to prove that you actually have control over the domain which will appear on the certificate and changes your NGINX configuration so that it encrypts and decrypt traffic with it.

Finally, since the application is going to be publicly available on the Internet, some sort of access control mechanism is necessary. Since this feature was not in the initial requirements, the system was not designed with that goal in mind. However, as a workaround, we can leveraged the fact that all traffic must go first through NGINX and implement rules which reject the requests which do not fulfill a certain condition. Specifically, NGINX includes a module which implements HTTP basic authentication scheme [40]. Simply put, this schema requires that every HTTP request contains an *Authorization* header with the access credentials encoded in base64. Therefore, when this module is enabled, NGINX only allows those requests which contain the correct credentials in the aforementioned header. The credentials are stored encrypted (i.e., hashed with a salt) on a file which is referenced in the NGINX configuration file.

In a browser, when a user tries to connect to the web application, a prompt is open which asks them for their credentials. Once entered, the browser sends them in every request which targets the web application origin, including the ones made by JavaScript using `fetch()`. The credentials are cached until the user closes the browser. In the XR application, the *Authorization* header has to be explicitly included in every call to the backend.

Serving static files

NGINX also specializes in serving static files and that is the reason why it is used for that task instead of FastAPI. The hardest part is getting the permissions of the static resources and upper directories right. NGINX requires that every file which is going to serve to be owned by the user `nginx` or belong to the group `nginx`. The problem is that the images are saved to the file system by a container, and as a consequence, the are owned by my user and belong to the group with the same name. A solution could have been changing the image so that

[40] `https://datatracker.ietf.org/doc/html/rfc7617`

the saved files belong to the `nginx` group. However, in my opinion, this is very convoluted to achieve. A simpler approach is to change the group of the directory where the static resources are saved to `nginx` and set its `setgid` bit. This causes every file created inside the directory to belong to the same group as the directory which contains that bit. Therefore, all files created by the container belong to the group `nginx`.

5.7 Design patterns

A design pattern is a general and reusable solution to a common problem in software design. It is a template or guideline that can be applied to solve a specific design issue in a particular context. Design patterns provide a proven approach to designing software systems, making it easier for developers to tackle recurring design challenges efficiently and effectively. A very influential book on this topic was the one written by the Gang of Four [GHJV94].

The most relevant design patterns which have appeared throughout this project are described below.

5.7.1 Python decorators with arguments

Python has a language feature called decorators [41]. Although it is called in the same way, it has little to do with the design pattern described in the Gang of Four book [GHJV94]. A decorator is actually syntactic sugar for wrapping a function to augment its functionality. Specifically, it provides new syntax for the following operation:

```
func = decorator(func)
```

where `func` is a function and `decorator` is another function which takes in a function and returns another function which accepts the same arguments as `func`. The syntax for decorators also support specifying arguments so that the function which wraps depends of those one. This is how it would look like if decorators were not used:

```
func = decorator_args(arg1, arg2)(func)
```

but with decorators, the relationship between `decorator_args` and `func` is stated on top of `func` definition by writing `@decorator_args(arg1, arg2)`

Decorators are the way in which handlers for a specific route in FastAPI are specified. The main advantage is that all the properties to configure the route are next to the definition of the function which is called when a request to that route is received.

[41] `https://realpython.com/primer-on-python-decorators`

5.7.2 Object pool

The object pool pattern is a creational design pattern that manages a pool of reusable objects to optimize performance and resource usage, especially when creating and destroying objects is costly in terms of time and system resources [GHJV94].

This pattern is used by SQLAlchemy. The library makes use of a pool of active connections to the database so that every time a query is needed to be sent, an old connection is reused instead of opening a new one. This is done primarily for performance reasons. However, in the case of MariaDB, if the connection has not been used to transmit data for more than eight hours, it is dropped without notifying SQLAlchemy. As a consequence, a stale connection could be retrieved to send the query, resulting in an error. The solution lies on configuring the connection pool to reset the connection after some period of time of inactivity lower than the maximum limit imposed by this database.

5.7.3 Iterator

The iterator pattern is a behavioral design pattern that provides a way to access the elements of a collection sequentially without exposing its underlying representation. This pattern allows a client to traverse a collection, element by element, without needing to know the details of how the collection is implemented [GHJV94].

This pattern is embedded at the language level in Python, TypeScript and C#. This is because the three have a `foreach` statement which allow the traversal of different structures without knowing lower level details about them by requiring them to adhere to a certain interface. Every time a `foreach` statement has been used, this pattern was implicitly used.

5.7.4 Adapter

The adapter pattern is a structural design pattern that allows objects with incompatible interfaces to work together. It acts as a bridge between two incompatible interfaces by converting the interface of a class into another interface that a client expects [GHJV94].

This pattern is heavily used by LlamaIndex. This is because one of the objectives of LlamaIndex is to provide a uniform interface to several vector database so that the library is not coupled to a single one. However, each vector database vendor defines its own way of interacting with it so LlamaIndex employs this pattern to make each one compatible with the interface they have defined.

Chapter 6

Results

In this chapter, it will be proven that the designed protocol editor is expressive enough for the practical case by specifying a real medical protocol. Additionally, the protocol will be augmented with images and documents containing relevant information about it. Then, it will be executed on the Meta Quest 3 headset using the developed MR application, where the attached images will be shown and the LLM will be queried to receive answers based on the uploaded documents.

The chapter finishes by giving some statistics about the written code. The source code is available under the MIT licence and is hosted on Github. The link to the repository is the following:

`https://github.com/AIR-Research-Group-UCLM/primARy`.

6.1 Real world example

In this part, the protocol is implemented and additional information is uploaded. Moreover, it is executed on the Meta Quest 3 headset.

6.1.1 Uploading information

The medical protocol which is going to be implemented is based on the one shown in figure 5.3, which was uploaded by the NIH to its webpage. It depicts the process of diagnosing and treating overweight.

The protocol is divided into two parts: one is the diagnosis, in which different steps are performed to determine if the patient has overweight and the other is the treatment.

The protocol starts by asking the patient whether they had ever a had a Body Mass Index (BMI) [1] greater than 25. If that has never happened, the last time the BMI was measured is asked. If it has been two years since that happened or, directly, the patient had a BMI greater than 25 in the past, the BMI is calculated. Additionally, measurements about the patient waist are also taken. Once it has been measured, the next steps are determined by the range in which that number falls. These are detailed below.

[1] BMI is a number calculated as a function of person's weight and height. The range in which that number falls determines whether they are underweight, overweight, obese or have an acceptable weight

- **BMI** $<$ **25**. In this situation, the previously measured circumference is compared with a number determined by the sex of the patient. If it is greater than that number, then overweight risk factors are assesed. If more than two risk factors are found, the patient is considered overweight. In case there are not enough risk factors, the patient can choose between starting a weight loss treatment or do nothing.
- **25** $\leq$ **BMI** $<$ **30**. Risk factors are directly assesed. The criteria for determining whether the patient has overweight given the number of risk factors is the same as the one described when BMI $<$ 25.
- **BMI** $\geq$ **30**. In this case, the patient is directly considered overweight.

The main protocol steps associated to the process described previously are shown in figure 6.1.

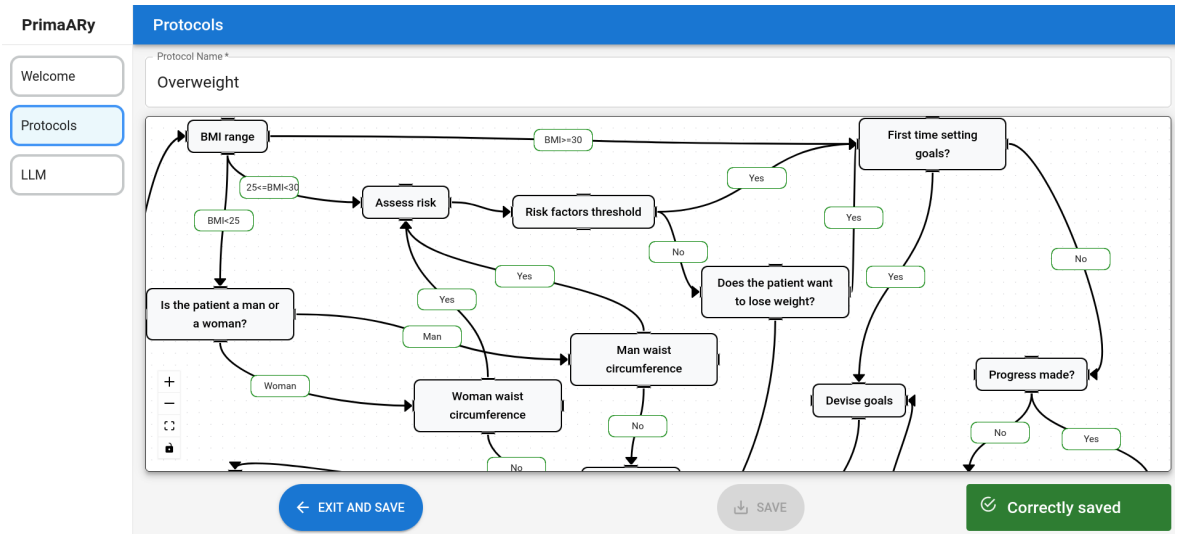

Figure 6.1: Overweight diagnosis and treatment protocol in the protocol editor

In case the patient has been diagnosed with overweight, the next steps describe the treatment. It basically consists in setting some goals regarding diet and exercise. If goals have previously been devised but the person is still considered overweight, the reasons for that failure are evaluated and new goals are set. The protocol always finishes by reminding the patient to schedule another appointment.

Moreover, some images have been attached to some steps. This is the case for the protocol step named *BMI range*. An image of the dialog containing the image is shown in figure 6.2

Finally, extra information has been gathered from the WHO website[2], the NHS website[3] and an article from Harvard[4] to provide context to the LLM. Since the system cannot extract

[2] `https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight`
[3] `https://www.nhs.uk/conditions/obesity/`
[4] `https://www.health.harvard.edu/staying-healthy/why-people-become-overweight`

information from an HTML file, the content of each webpage has been copy-pasted into a `.txt` file and associated to the protocol. These can be seen in figure 6.3.

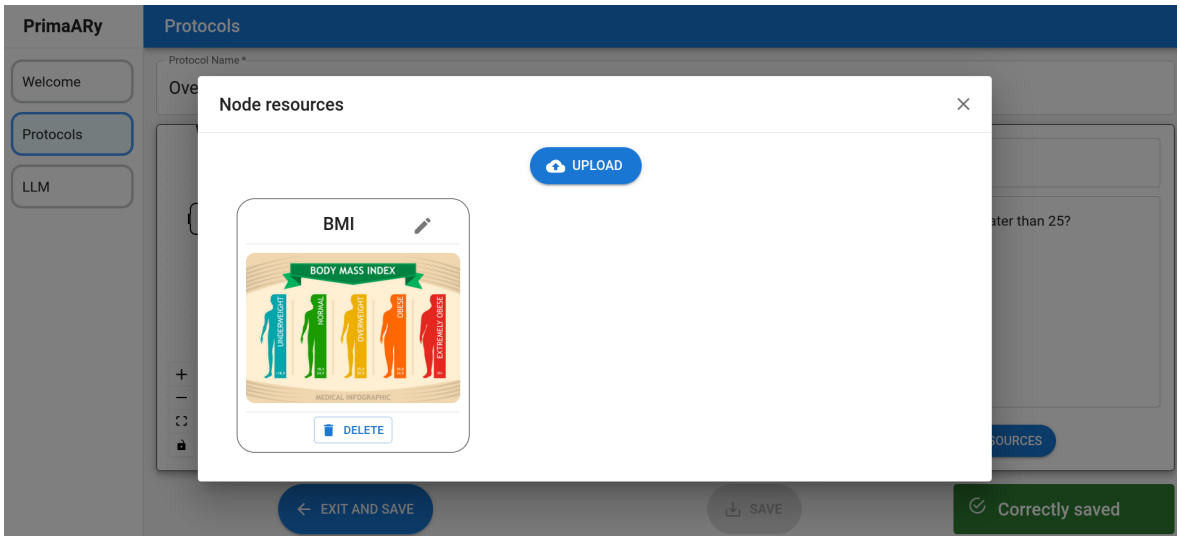

Figure 6.2: Image associated to protocol step *MBI range*

Figure 6.3: Dialog showing documents containing information about overweight

6.1.2 Asking questions to the LLM

In order to test whether the LLM is using the information provided in the three documents uploaded previously, questions whose answer is unlikely to be known by it are going to be asked. The model has been configured with a context window of 8192 tokens. Moreover, the number of chunks which are retrieved from vector database are six.

One question is *How many genes are thought to be related to overweight?*. The answer is 400 and it is found in the Harvard article. Another question which has been asked is *What daily problems a person with obesity might experience?*. Its answer is located on

the document extracted from the NHS. Some problems include breathlessness, increased sweating or low confidence and self-esteem.

The question using the concatenate generation mode are shown in figure 6.4. As it can be seen, it has responded correctly.

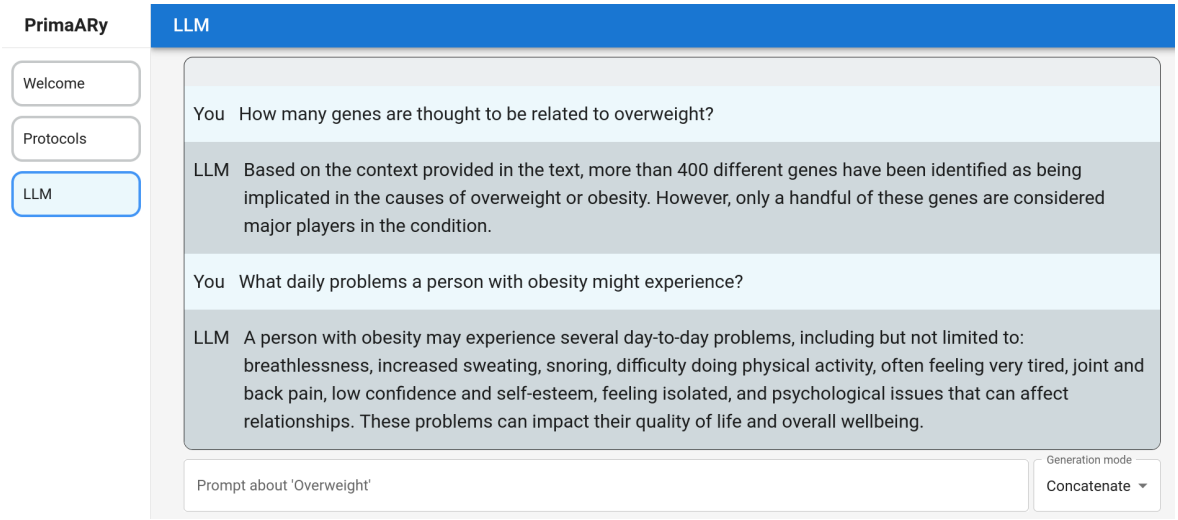

Figure 6.4: Answers by the LLM using concatenate mode (six chunks)

In multi-step mode, which is supposed to be more accurate, the model gets the answer right in the first paragraph. However, it includes much more superfluous information. So much, that the given answer consists of four paragraphs. In fact, the last one finishes with the economic impact of obesity, which is not relevant in this case. For the other question, the model also gives the correct information but among irrelevant facts about obesity. Additionally, it has hallucinated the reasons which cause the problems because that information is not found in the provided documents.

Since in multi-step generation mode the model does not seem capable of knowing whether a previous response contains relevant information, the number of retrieved chunks was reduced from six to three. The quality of the results improved considerably. The model did not hallucinate this time and its responses were more aligned with the posed questions.

6.1.3 Executing a protocol on the Meta Quest 3

When the user puts on the Meta Quest 3, they can see their environment thanks to the pass-through feature. When they turn their left hand so that their palm is in front of them, a list of the protocols created with web application appear. This list is shown in figure 6.5. The application performs hand tracking, so it is enough to make the gesture of pressing a button in order to launch it.

After it is launched, a floating canvas containing the title and description of the initial step appears. At the bottom of the canvas, there are as many buttons as outgoing edges the initial step in the flowchart has. These represent the possible options the user can choose. In

case there is just one outgoing edge, a single button with the text *Continue* is shown. After the user presses a button, the canvas will be updated to show the information corresponding to the next step and the interactions with it will be the same. In case it is a final step, the available options are retrying the protocol or closing it. The user can, at every moment, go to a previous state by pressing the button located in the upper left corner of the canvas, except for the initial step.

Figure 6.5: List of protocols as displayed by the XR application

For every step, there is a button that, when pressed, shows a canvas which allows you to ask questions about the protocol it is being executed. The questions are asked out loud and the model responds using a text-to-speech model. This canvas has a dropdown menu for choosing the generation mode. For every step which have images attached, an additional button appears which shows a third canvas with those images when it is pressed. The three canvases are shown in figure 6.6 for the protocol step named *BMI range* in the protocol depicted in figure 6.1.

All those canvases can be freely moved by grabbing them. However, they will always be looking at the user so that their content is easily visible.

6.2 Code statistics

The lines of code of each component of the system is given in table 6.1. The tools which has been used is `cloc`[5].

System component	Lines of code	Percentage
LLM service	362	7%
Protocol service	815	15.6%
Web application	3337	63.8%
Quest 3 application	713	13.6%
	5227	100 %

Table 6.1: Lines of code of each system component

[5] `https://github.com/AlDanial/cloc`

Figure 6.6: Step named *BMI range* shown on the XR application

When looking at data in table 6.1, one has to be aware of the fact that not all the components of the system have been written in the same programming languages. There exist some which are more verbose because they require more code to express the same equivalent behaviour.

I personally think that it is important to have an idea of the lines of code a codebase has because it is an indicative of its complexity. Nevertheless, it is important not to get obsessed with the quantity because some people, in effort to make code shorter, tend to write it in such a way that is harder to read than its most straightforward alternative.

6.3 Project cost and resources

In order to estimate the cost of this project, the time it has taken to be developed has to be taken into account. Considering that I have worked part-time on it while completing the fourth year , it has been estimated that 270 hours have been dedicated exclusively to the development of the system. If the time spent writing this document is also considered, then the total time amounts to 325 hours.

The hourly salary for a programmer is 14.62€[6], so the total gross salary I have received is $270 \times 14.62 = 3,947.3$€. However, this is not the total personnel costs of the company because it has to pay the social security contribution which correspond to it. This amounts to, approximately, 30% of the gross salary. Thus, the total personnel costs for the company has been 5,131.49€.

I am being charged 9.68€ per month for the IONOS VPS Linux M[7] because it was hired more than six months ago. The project lasted four months so the VPS has cost 38.72€ in total. Additionally, one does not have to forget the cost of the domain name. Since it was a non-premium domain and it was the first time I rent a domain from IONOS, I had to pay 1 € only once. If I want keep it, I will have to pay 10€ per year.

Other hardware resources which have been employed for the development of the project include the Meta Quest 3 and my PC. The former cost 549€ and the latter was bought with a price of 1,300€. Additionally, the amortized cost of the server where the LLM is running has been estimated, resulting in 266.66€.

Therefore, the total cost of the project amounts to 5,131.49€ + 38.72€ + 1€ + 549€ + 1,300€ + 266.6€ = 7,286.87€.

[6] https://es.talent.com/salary?job=programador

[7] https://www.ionos.es/servidores/vps

Chapter 7

Conclusions

In this chapter, the reached objectives will be discussed. Following that, the addressed competencies are detailed and justified in the context of this project. Then, a personal opinion of the project is given. Finally, future work and possible improvements to the developed system are outlined.

7.1 Reached objectives

The general objective has been successfully fulfilled. This is because the developed system enhances the diagnostic and treatment capabilities of primary care medical stuff using MR and AI by enabling them to specify the medical protocols they usually follow in the form of a flowchart and executing them on a MR headset like the Meta Quest 3. The AI component is provided by the LLM, which synthetizes a response considering previously uploaded documentation by the protocol designer.

Below, the subgoals in which the project was decomposed are detailed with an explanation justifying their degree of achievement.

- **Detailed study of Mixed Reality devices.** In order to develop the MR application, these kind of devices were researched as well as the tools used to create applications. Part of the acquired knowledge has been reflected on section 3.3.1 and section 5.5.1. An exhaustive comparison between MR devices was not performed because the application was targeted since the beginning to the Meta Quest 3 and it is not compatible with headsets from other companies because vendor specific OpenXR extensions were used.

- **Detailed study of Large Language Models.** A study of the field of deep learning in general and LLMs in particular was performed to see how they could be used to enhance the diagnosis capabilities of primary care staff. That resulted in the incorporation of a RAG process to include the information contained in documents into the answers of the model. Additionally, the study of the techniques used to run a LLM on consumer hardware lead to the appropriate software tools which enabled self-hosting one. All of that information appears in subsections 3.1.1 and 5.4.4.

- **Design and development of a web application that allows the definition of medical protocols and the association of information which serve as a basis for the inference process.** This specific objective has been fulfilled since a flowchart editor was developed with the necessary features to easily specify a medical protocol. Moreover, it includes functionality which allows the user to attach documents to a protocol.

- **Design and development of a MR application that allows the execution of medical protocols by primary care staff**. An application for the Meta Quest 3 which executes the protocols specified in the web application has been successfully developed. It also enables the medical staff to interact with the LLM by talking to it and the model will read the generated response out loud.

- **Deployment of a real medical protocol which allows the diagnosis and treatment of a disease.** The medical protocol for overweight diagnosis uploaded by the NIH to its webpage (see figure 5.3) was specified using the web application and successfully executed on the application designed for the Meta Quest 3. Additionally, the responses given by the LLM to the posed questions were decent considering that it is a small LLM.

7.2 Addressed competences

In this section, the addressed competencies will be outlined.

- **[CM4] Ability to understand the fundamentals, paradigms and techniques of intelligent systems and to analyze, design and build systems, services and computer applications that use these techniques in any field of application.** The developed system makes use of a LLM which acts as an assistant which helps answering the questions of the professional when they are executing a protocol. Additionally, the protocol itself could be considered an agent which guides the professional towards the right diagnosis or treatment given the information provided by them while executing it. In order to develop each intelligent part of the system and combine their functionality, deep knowledge about the techniques which make intelligent system possible is required.

- **[CM6] Ability to develop and evaluate interactive and complex information presentation systems and their application to the resolution of human-computer interaction design problems.** One of the core parts of the developed system is the MR application in which protocols are executed. Presenting that type of information using the kinds of interactions which characterizes the MR requires a deep knowledge of the paradigm so that the resulting system is usable. On the other hand, the UI developed to create the protocols is non-trivial since all the relationships between steps in a protocol have to be specified in a easy way to ensure the usability of the application.

7.3 Personal conclusion

This project is the culmination of my Computer Engineering degree and it would not have been possible without the skills and knowledge acquired during the four years it lasted.

The development of this undergraduate project has been a great opportunity to dive deeper into the fields of XR, LLMs and web development. Specifically, before starting this project I have never tried an XR headset and, as a consequence, I did not know much about the technology. However, working with the Meta Quest 3 made me realize that this paradigm of interaction has a lot of potential in areas such as medicine or e-commerce. Although it is still a niche field, I suspect that when wearing a device is as easy as wearing a pair of glasses, the technology will be a total game-changer in a lot of economic activities and in the way we interact with each other.

On the other hand, including a LLM into the developed system has been one of the best incentives to better understand the kinds of models which power systems such as ChatGPT. Even though I personally think that LLMs are not the path which will lead us to Artificial General Intelligence (AGI), I believe, nevertheless, that they will be a core component in advanced intelligent systems which will help us in ways that we are just starting to glimpse. Moreover, it has been really satisfying to be able to run a LLM locally. I strongly believe that in order for people to fully benefit from the capabilities offered by LLMs, the will have to be able to execute them privately and without the intervention of big companies.

Additionally, this project has given me the necessary confidence to tackle other software systems involving web application. Frontend development has been specially harder than I initially thought and the great amount of tools there exist to architect a frontend as well as the fast-pacing nature of this field does not make it any easier. Nevertheless, the initial confusion gradually started to disappear as I learnt more concepts and applied them to the web application.

Overall, although there is room for improvement, I am happy with the results.

7.4 Future work

In this section, improvements and future work is discussed.

- **Test the system on a real setting**. Once a functional prototype is built, the next phase consists in validating it with the people for whom it is targeted: primary care medical staff. A lot of feedback could be received from them which would help improve the system.
- **Support for multiple users**. The system is designed to be used just for a person. If other professionals used the system, their protocol will be intermingled with those specified by others. Moreover, I would like to scale the LLM service so that it handles multiple concurrent queries instead of responding with a server-side error.

- **Support for attaching other media files to a protocol step**. It is only possible to attach images to a specific protocol step. However, it would be a good idea to also be able to link videos or even 3D models. The difficulty of the former lies on the fact that, in order to provide a good UX, the video should be streamed and, with the other, is that dynamically loading 3D in Unity is not as straightforward as one might think initially.
- **Save statistics about an executed protocol**. Different kinds of statistics could be saved about the execution of a protocol so that they can be analyzed to provide insight about one or a group of them.
- **Explore alternatives to LlamaIndex**. LlamaIndex is a good library for fast prototyping. However, it abstracts too much details and it has been very hard to figure out how it was processing several things. Moreover, its inflexibility has determined part of the system architecture, as explained in section 5.6. Given that LlamaIndex was not doing anything very sophisticated to preprocess the data, I think that the advantages of implementing a RAG process from scratch outweigh the disadvantages.
- **Explore the possibility of using more kinds of interactions offered by Meta Quest 3**. The headset which has been used for the development of the application provides more MR functionalities apart from the ones which has been used, such as eye tracking, face tracking or support for audio spatialization. Exploring them in more depth could improve the application UX.
- **Remove the need of creating *repeated* steps to remember the chosen option**. Suppose that you have a step which tells the professional to ask whether the patient suffers from a specific condition. If you want to remember that decision in another step, the technique is to ensure that to reach it, all possible paths from the initial step to that one includes the edge containing that option. However, that may lead to two steps containing exactly the same information with the only difference being that one could have only be reached if the user chose one option and the other if they did the same with the other. The flowchart editor could be adapted to avoid that *duplicated* node.

Appendices

Appendix A

A.1 Instructions for running the system

This section explains how to configure the system in order to execute it. The configuration files have been designed for the setup outlined in section 5.6, where some services are running on a machine and the others in a different one. However, the procedure to adapt it for other setups is explained.

A.2 Requirements

Docker needs to be installed on the system. Rootless Docker is recommended if you do not want to type `sudo` every time you invoke a Docker command. Although I thought that it could be very difficult to get it working, it turned out to be as simple as working without it, at least for this case.

Additionally, since the LLM service make use of GPU, two packages have to be installed on the system. In order for a container to use a NVIDIA GPU, the packages `nvidia-cuda-toolkit` [1] and `nvidia-container-toolkit` [2] have to be installed on the Docker host. There are additional instructions in case you have opted for rootless Docker.

A.2.1 Configuring the services

For my specific setup, there exist two `compose.yml` files because a set of services are running on the VPS and the rest are being executed on *airproy*. One is located at the root directory of the project and the other is found inside the `llm` directory. However, if you want to run all the services on a single machine you can copy and paste the services defined in the `compose.yml` found in `llm` to the other one. The only detail that needs to be changed is the build context for the LLM service. You do that by replacing `context:.` with `context: ./llm`.

The services are configured using environment variables whose values are specified in the `compose.yml` file where the service is defined. A list of those environment variables can be seen in section A.3. Moreover, Docker port forwarding should be configured appropriately

[1] `nvidia-cuda-toolkit`

[2] `https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html`

depending on whether the services are required to be directly accessed from outside the Docker network created by Docker Compose.

To run all the services defined in a `compose.yml` file, make the directory where it is found your working directory (i.e., `cd` into it) and type `docker compose up -d`. To stop them, type `docker compose down`.

A.3 Environment variables

Here the environment variables of each service are listed. Each one has sensible defaults, except for API_BASE and LLM_SERVICE. Those have to be explicitly set by the user. Additionally, for a production setup, I recommend serving the static content using a web server designed for that task, such as NGINX. In that case, STATIC_BASE must be set appropriately.

A.3.1 LLM service

- **CONTEXT_WINDOW**. Context length configured for the model.
- **N_GPU_LAYERS**. Number of layers which are offloaded to the GPU
- **MAX_TOKENS**. Maximum number of tokens the model generates, including the ones from the prompt.
- **QDRANT_HOST**. Vector database origin. For example, `http://192.168.0.10:3000`. You should not need to assign a value unless you are running the vector database outside the Docker network created by Docker Compose.
- **MODEL_PATH**. Path to the file containing the model files. You should not need to assign this value unless you want to use a version of the weights than the one contained in the Docker image.
- **DEFAULT_TOP_K**. Default number of chunks which are retrieved from the vector database.
- **IS_DEV**. It can be 0 or 1. When it equals 1, CORS headers are added which allow the service to be contacted from all origins.

A.3.2 Protocol service

- **LLM_SERVICE_HOST**. Host in which the LLM service is running.
- **DB_HOST**. Host where MariaDB is running. You should not need to assign a value unless you are running the vector database outside the Docker network created by Docker Compose.
- **DB_USERNAME**. User which is going to be used to connect to the database.
- **DB_PASSWORD**. Password of the user connecting to the database.

- **DB_PORT**. The port MariaDB is listening to.
- **STATIC_PATH**. The route where all static files live. By default, it is set to `/static`.
- **IS_DEV**. It can be 0 or 1. When it equals 1, When it equals 1, CORS headers are added which allow the service to be contacted from all origins and static files are served using FastAPI. This is very useful for development purposes.

A.3.3 NextJS service

- **API_BASE**. URL prefix of all the protocol service API endpoints without the trailing slash. For example, `https://example.es/api`.
- **LLM_BASE**. URL prefix of all LLM service API endpoints without the trailing slash. For example, `https://example.es/api/llm`.
- **STATIC_BASE**. Prefix of all the URLs which point to static resources. For example, `https://example.es/static`

References

[Ack23] Philip Ackermann. *Full Stack Web Development: The Comprehensive Guide*. Rheinwerk Computing, 2023.

[AMHH19] Tomas Akenine-Moller, Eric Haines, y Naty Hoffman. *Real-time rendering*. AK Peters/crc Press, 2019.

[AMK21] Firas Almukhtar, Nawzad Mahmoodd, y Shahab Kareem. Search engine optimization: a review. *Applied computer science*, 17(1):70–80, 2021.

[BPRS18] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, y Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey, 2018.

[CG23] Deniz Akşimşek Carson Gross, Adam Stipenski. *Hypermedia Systems*. Independently published, 2023. `https://hypermedia.systems/`.

[Cha21] Stanley H. Chan. *Introduction to Probability for Data Science*. Michigan Publishing Services, 2021. `https://probability4datascience.com/`.

[CJA+20] Umer Asghar Chattha, Uzair Iqbal Janjua, Fozia Anwar, Tahir Mustafa Madni, Muhammad Faisal Cheema, y Sana Iqbal Janjua. Motion sickness in virtual reality: An empirical evaluation. *IEEE Access*, 8:130486–130499, 2020.

[Cra13] Alan B. Craig. Chapter 7 - Mobile Augmented Reality. En Alan B. Craig, editor, *Understanding Augmented Reality*, páginas 209–220. Morgan Kaufmann, Boston, 2013. url: `https://www.sciencedirect.com/science/article/pii/B9780240824086000072`.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, y Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DNBM12] Torgeir Dingsøyr, Sridhar Nerur, VenuGopal Balijepally, y Nils Brede Moe. A decade of agile methodologies: Towards explaining agile software development, 2012.

[DR15] Joydip Dhar y Ashok Ranganathan. Machine learning capabilities in medical diagnosis applications: Computational results for hepatitis disease. *International Journal of Biomedical Engineering and Technology*, 17(4):330–340, 2015.

[Els05] Amy Elser. *Reliable distributed systems: technologies, web services, and applications*. Springer Science & Business Media, 2005.

[FAHA22] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, y Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

[GBC16] Ian Goodfellow, Yoshua Bengio, y Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GGI20] Alina-Mădălina Gheorghe, Ileana Daniela Gheorghe, y Ioana Laura Iatan. Agile Software Development. *Informatica Economica*, 24(2), 2020.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, y John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, edición 1, 1994. url: `http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1`.

[GSK+19] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, y Rogerio Feris. Spottune: transfer learning through adaptive fine-tuning. En *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, páginas 4805–4814, 2019.

[GXG+23] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, y Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

[Hav18] Marijn Haverbeke. *Eloquent javascript: A modern introduction to programming*. No Starch Press, 2018.

[HBD+19] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, y Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[HFS+19] Hong-zhi Hu, Xiao-bo Feng, Zeng-wu Shao, Mao Xie, Song Xu, Xing-huo Wu, y Zhe-wei Ye. Application and prospect of mixed reality technology in medical field. *Current medical science*, 39:1–6, 2019.

[Hig13] Jim Highsmith. *Adaptive software development: a collaborative approach to managing complex systems*. Addison-Wesley, 2013.

[HLW23] Yikun Han, Chunjiang Liu, y Pengfei Wang. A comprehensive survey on vector database: Storage and retrieval technique, challenge. *arXiv preprint arXiv:2310.11703*, 2023.

[HM08] Allen L Hixon y Gregory G Maskarinec. The Declaration of Alma Ata on its 30th anniversary: relevance for family medicine today. *Fam Med*, 40(8):585–8, 2008.

[HNM16] Samuel Heuts, Peyman Sardari Nia, y Jos G Maessen. Preoperative planning of thoracic surgery with use of three-dimensional reconstruction, rapid prototyping, simulation and virtual navigation. *Journal of Visualized Surgery*, 2, 2016.

[HQS+23] Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. *Authorea Preprints*, 2023.

[HTFF09] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, y Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[JK18] Lasse Jensen y Flemming Konradsen. A review of the use of virtual reality head-mounted displays in education and training. *Education and Information Technologies*, 23:1515–1529, 2018.

[Jon17] Capers Jones. *Software methodologies: a quantitative guide*. Auerbach Publications, 2017.

[Kar16] Andrej Karpathy. *Connecting images and natural language*. Phd thesis, August 2016. Available at `https://cs.stanford.edu/people/karpathy/main.pdf`.

[KBH+15] Dionne S Kringos, Wienke GW Boerma, Allen Hutchinson, Richard B Saltman, World Health Organization, et al. *Building primary care in a changing Europe*. World Health Organization. Regional Office for Europe, 2015.

[Kle17] Martin Kleppmann. *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. " O'Reilly Media, Inc.", 2017.

[LC17] Fernando Lamata Cotanda. Atención Primaria en España: Logros y Desafíos. *Revista Clínica de Medicina Familiar*, 10:164 – 167, 10 2017. url: `http:`

//scielo.isciii.es/scielo.php?script=sci_arttext&pid=S1699-695X2017000300164&nrm=iso.

[MBHN24] Nicolo Micheletti, Samuel Belkadi, Lifeng Han, y Goran Nenadic. Exploration of Masked and Causal Language Modelling for Text Generation. *arXiv preprint arXiv:2405.12630*, 2024.

[MMN+24] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, y Jianfeng Gao. Large language models: A survey. *arXiv preprint arXiv:2402.06196*, 2024.

[Mol20] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[Org23] World Health Organization. Primary health care, 2023. https://www.who.int/news-room/fact-sheets/detail/primary-health-care [Accessed: 25 June 2024].

[PH22] Mary Phuong y Marcus Hutter. Formal algorithms for transformers. *arXiv preprint arXiv:2207.09238*, 2022.

[PMB22] Adéla Plechatá, Guido Makransky, y Robert Böhm. Can extended reality in the metaverse revolutionise health communication? *NPJ digital medicine*, 5(1):132, 2022.

[RBR15] Philipp A Rauschnabel, Alexander Brem, y Young Ro. Augmented reality smart glasses: definition, conceptual insights, and managerial importance. *Unpublished Working Paper, The University of Michigan-Dearborn, College of Business*, 2015.

[RH05] Jannick P Rolland y Hong Hua. Head-mounted display systems. *Encyclopedia of optical engineering*, 2:1–14, 2005.

[Rod15] Carlos Cuadros Rodríguez. Modelo de Ayuda al Diagnóstico de Infecciones del Tracto Urinario en la Atención Primaria de Salud, July 2015.

[SC03] William R Sherman y Alan B Craig. Understanding virtual reality. *San Francisco, CA: Morgan Kauffman*, 2003.

[SEK24] Harald Steck, Chaitanya Ekanadham, y Nathan Kallus. Is Cosine-Similarity of Embeddings Really About Similarity? En *Companion Proceedings of the ACM on Web Conference 2024*, WWW '24. ACM, Mayo 2024. url: http://dx.doi.org/10.1145/3589335.3651526.

[Sev12] Charles Severance. JavaScript: Designing a language in 10 days. *Computer*, 45(2):7–8, 2012.

[SSC+18] Joshua Warren Sappenfield, William Brit Smith, Lou Ann Cooper, David Lizdas, Drew B Gonsalves, Nikolaus Gravenstein, Samsun Lampotang, y Albert R Robinson III. Visualization improves supraclavicular access to the subclavian vein in a mixed reality simulator. *Anesthesia & Analgesia*, 127(1):83–89, 2018.

[TSK+20] Amirsina Torfi, Rouzbeh A Shirvani, Yaser Keneshloo, Nader Tavaf, y Edward A Fox. Natural language processing advancements by deep learning: A survey. *arXiv preprint arXiv:2003.01200*, 2020.

[TYLG+19] Shuo Tian, Wenbo Yang, Jehane Michael Le Grange, Peng Wang, Wei Huang, y Zhewei Ye. Smart healthcare: making medical care more intelligent. *Global Health Journal*, 3(3):62–65, 2019.

[VRW+16] Joost Visser, Sylvan Rigal, Gijs Wijnholds, Pascal Van Eck, y Rob van der Leek. *Building Maintainable Software, C# Edition: Ten Guidelines for Future-Proof Code*. " O'Reilly Media, Inc.", 2016.

[WS12] Shijun Wang y Ronald M Summers. Machine learning and radiology. *Medical image analysis*, 16(5):933–951, 2012.

[WSZ+22] Yuntao Wang, Zhou Su, Ning Zhang, Rui Xing, Dongxiao Liu, Tom H Luan, y Xuemin Shen. A survey on metaverse: Fundamentals, security, and privacy. *IEEE Communications Surveys & Tutorials*, 25(1):319–352, 2022.

[ZLLS23] Aston Zhang, Zachary C. Lipton, Mu Li, y Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. `https://D2L.ai`.

Este documento fue editado y tipografiado con LaTeX empleando
la clase **esi-tfg** (versión 0.20181017) que se puede encontrar en:
`https://bitbucket.org/esi_atc/esi-tfg`